MASTER EN SCIENCES INFORMATIQUES

Analyse et applications des réseaux de Petri temporels

Hutlet, Marc

Award date:
1990

Awarding institution:
Universite de Namur

Facultés Universitaires N.D. de la Paix, Namur
Institut d'Informatique
Rue Grandgagnage, 21
B - 5000 NAMUR (Belgium)

Année académique 1989-1990

ANALYSE ET APPLICATIONS DES RESEAUX DE PETRI TEMPORELS

Marc HUTLET

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique

Titre

Analyse et Applications des Réseaux de Petri Temporels

Résumé

Ce mémoire est consacré à l'étude d'un modèle formel permettant d'exprimer explicitement le temps, pour la spécification et la vérification des systèmes distribués.

Le modèle utilisé est le modèle des réseaux de Petri temporels de Merlin.

Une approche par énumération est introduite pour l'analyse de ces réseaux. Elle est basée sur le calcul d'un ensemble de classes d'états et sur la définition d'une relation d'accessibilité sur cet ensemble.

Les propriétés principales du modèle sont analysées et certaines limites de la méthode d'analyse par énumération sont mises en évidence.

Ensuite, l'élaboration d'un logiciel implémentant la méthode d'énumération est présentée et les caractéristiques d'un outil professionnel sont détaillées. De tels outils facilitent l'édition de systèmes complexes et permettent une meilleure analyse de ces systèmes.

Une méthodologie d'analyse adaptée aux protocoles de communication est alors donnée et des applications sont traitées. Ceci contribue à montrer l'utilité des réseaux de Petri temporels pour la représentation et la vérification de protocoles de communication.

Finalement, les réseaux de Petri temporels sont appliqués à l'analyse de la couche liaison de données du protocole FIP (Flux Information Processus) utilisé dans les processus de production automatisés.

Mots-clés

Réseaux de Petri Temporels, Approche par Enumération, Spécification et Modélisation, Analyse et Vérification, Protocoles de Communication, Temps-limites.

Abstract

This thesis deals with the study of a formal model including time for the specification and verification of distributed systems. The model used is Merlin's time Petri nets.

An enumerative analysis technique is introduced for these nets. This technique is based on the computation of a set of state classes and on the definition of a reachability relation on the set.

The principal properties of the model are analysed and some limitations of the enumerative method are discussed.

Afterwards, the elaboration of a software which implements the enumerative method is proposed and a professional tool is detailed. Such tools provide facilities for editing complex nets and processing the results of analysis.

A methodology for analysing communication protocols is then introduced and applications are considered. They give the analysis possibilities of time Petri nets and show how these nets can be used to model and verify communication protocols.

Finally, time Petri nets are applied to the verification of the data link layer of the FIP protocol (Flux Information Processus), used in automatisated production processes.

Key-words

Time Petri Nets, Enumerative Approach, Specification and Modelling, Analysis and Verification, Communication Protocols, Time-outs.

AVANT-PROPOS

Le travail présenté dans ce mémoire a été réalisé en vue de l'obtention du diplôme de Licencié et Maître en Informatique.

Il clôture le cycle de cinq années d'études effectuées au sein des Facultés Universitaires Notre-Dame de la Paix à Namur.

Je tiens, en particulier, à exprimer toute ma gratitude à Monsieur P. Azéma pour m'avoir accueilli comme stagiaire au sein de l'équipe Outils et Logiciels pour la Communication (OLC) du Laboratoire d'Automatique et d'Analyse des Systèmes du Centre National de la Recherche Scientifique de Toulouse (LAAS), dirigé par Monsieur A. Costes que je remercie également pour son accueil.

Je tiens également à exprimer mes remerciements à tous les membres du groupe OLC pour leur soutien amical.

Je tiens à remercier :

- mon directeur de mémoire, Monsieur J. Fichefet,
- tous les membres du jury qui ont participé à ma soutenance,

ainsi que toutes les personnes qui de près ou de loin m'ont soutenu durant ces cinq années d'études.

E R R A T A

p.7 paragraphe 1.3, 4eme ligne: remplacer " elle " par " Elle "

p.10 paragraphe 2.2, 2eme ligne: remplacer " spéculations " par " spécifications "

2eme ligne du bas: remplacer " L_1 L_2 ...L_m <==> R " par " L_1 & L_2 &...& L_m <==> R "

1ere ligne du bas remplacer " L_1 L_2 ...L_m <==> R " par " L_1 & L_2 &...& L_m ==> R "

p.11 paragraphe 2.3.2, 3eme ligne: remplacer " R " par " Rθ "

8eme ligne du bas: remplacer " L_m R " par " L_m ==> R "

5eme ligne du bas: remplacer " <==> " par " ==> "

p.12 4eme ligne: remplacer " i " par " $\forall$ i "

9eme ligne: remplacer " $\{\phi$ " par " $\{\varphi\}$ " et remplacer " θ_n " par " θ_r "

p.14 5eme ligne: remplacer " de l'instanuation " par " l'instanciation"

7eme ligne: remplacer " une même variable " par " à une même variable "

10eme ligne: remplacer " de: " par " de &: "

17eme ligne: remplacer " à gauche " par " de gauche "

19eme ligne: remplacer " de jointures " par " de jointure "

p.15 paragraphe 2.4.5.1, 2eme ligne: remplacer " de jointures " par " de jointure "

5eme ligne: remplacer " instasuation " par " instanciation "

paragraphe 2.4.5.2, 4eme ligne: remplacer " approprié est mémorisé' par " appropriée est mémorisée "

p.16 12eme ligne du bas: " phase 2-3 (...) " doit venir à la ligne, aligné sur " fin si "

5eme ligne du bas: remplacer "L1" par "L_1"

p.17 12eme ligne du bas: remplacer "associée" par "associé"

9eme ligne du bas: remplacer "j" par "$\exists$ j"

p.19 paragraphe 3.1, 2eme ligne: remplacer "régle" par "règles"

7eme ligne: remplacer "stuctures" par structures"

paragraphe 3.2, 4eme ligne: remplacer "instanuation" par "instanciation"

1ere ligne du bas: remplacer " contrainte" par "contraintes"

p.20 paragraphe 3.3.2, 3eme ligne: remplacer "algorithme" par "algorithme:"

8eme ligne: remplacer "navigation" par " de navigation"

p.21 2eme ligne: remplacer "suivant" par "suivants"

p.22 instruction right-merge : lire " n_{2n}" au lieu de "n_{2m}" , "n_{2n-1}" au lieu de "n_{2m-1}" , "n_{21} " au lieu de "n_{2i}" et "n_{11}" au lieu de "n_{i1}"

instruction test-eq, 3eme ligne: remplacer "de ces littéraux" par "des littéraux"

5eme ligne: remplacer "littéraeux" par "littéraux"

p.23 3eme ligne: remplacer "P_{Pn1}" par "P_{PN1}" et "P_{Pn2}" par "P_{PN2}"

9eme ligne: remplacer "PPN2" par "P_{PN2}"

15eme ligne: remplacer "DIr_0" par DIR_0"

p.24 3eme ligne: remplacer "n_{1n+1}" par "n_{1m+1}"

remplacer "avec n_{1m+1} =..." par avec "n_{1m+1}= n_{3r} n_{2n}'"

p.25 7eme ligne: remplacer "arg_3 < arité" par " arg_3 <= arite"

instructions return et fail: inverser la pile d'adresses dans pré et post

p.26 la description de l'instruction fork a été accidentellement omise; insérer avant l'instruction stop le texte suivant:

fork un argument arg_1 de type "adresse d'instruction"

Pré: pile d'adresses = adr_p . . . adr_1

Post: pile d'adresses = next où "next" est l'adresse de l'instruction suivant le fork

adr_p . . . adr_1

L'éxécution se poursuit en arg_1

p.30 paragraphe 3.4.2, 9eme ligne: aligner "terminal0" sur le mot "instruction" de la ligne précédente

p.32 1ere et 2eme lignes: remplacer "noeud de filtrage" par "noeud terminal"

p.33 1ere ligne: remplacer "C_i" par "c_i"

15eme ligne et 16eme ligne: remplacer le texte par

"X >= Y & Y >= 0 ==> X >= 0

Y >= X & Y = 0 <==> X = 0"

8eme ligne du bas: remplacer " chanfement" par "changement"

7eme ligne du bas: remplacer " instan ant" par "instanciant"

p.34 paragraphe 3.4.6, 3eme ligne: mettre "p(_g406,vrai,_g406) <==> true" à la ligne

7eme ligne: mettre "q(a,Hobbit) <==> peregrin_took" à la ligne

p35 16eme ligne: supprimer la ligne

20eme ligne: remplacer "X" par "x"

p.37 7eme ligne: supprimer la ligne

p.40 paragraphe 4.1, 4eme ligne: remplacer "oar" par "par"

p.43 référence (4) remplacer "Intriduction" par "Introduction"

référence (7), 2eme ligne: remplacer " fort" par " for"

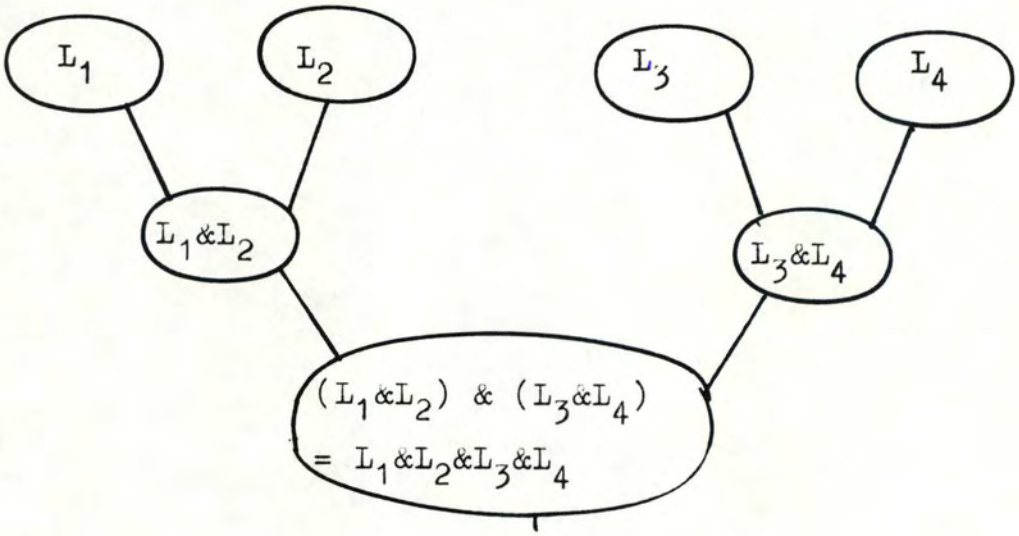
référence (8), 2eme ligne: remplacer "8eme journée" par "8emes journées"

p.46 7eme ligne du bas (IMPLIES): remplacer <==> par ==>

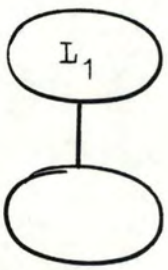
p.47 paragraphe 2.3, 9eme ligne: remplacer "intégrer" par "integer"

7eme ligne du bas: remplacer " argument" par "arguments"

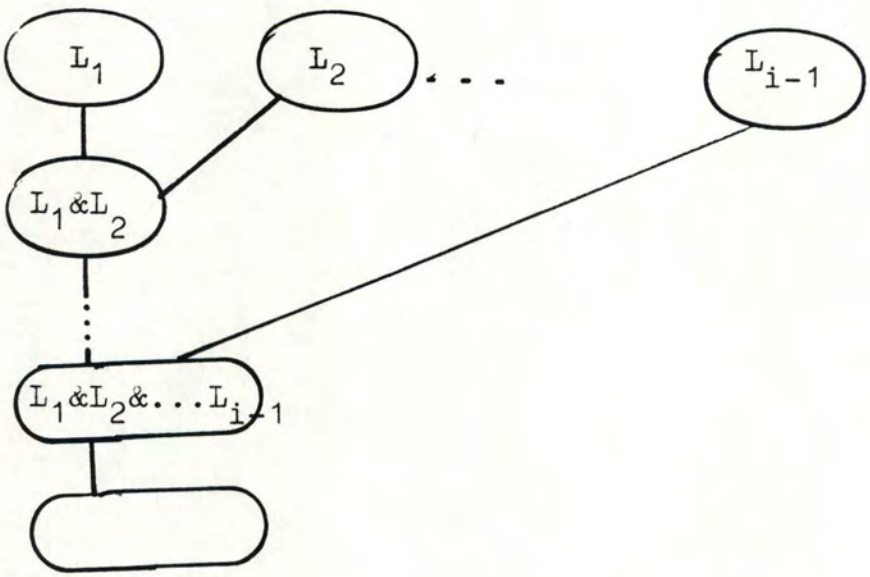
p.29 remplacer la figure 5 par



p.31 remplacer la figure du point a) par



remplacer la figure du point b) par



p.32 Iere et 2eme lignes: remplacer "noeud de filtrage" par " noeud terminal"

Insérer sous le point c) la figure suivante:

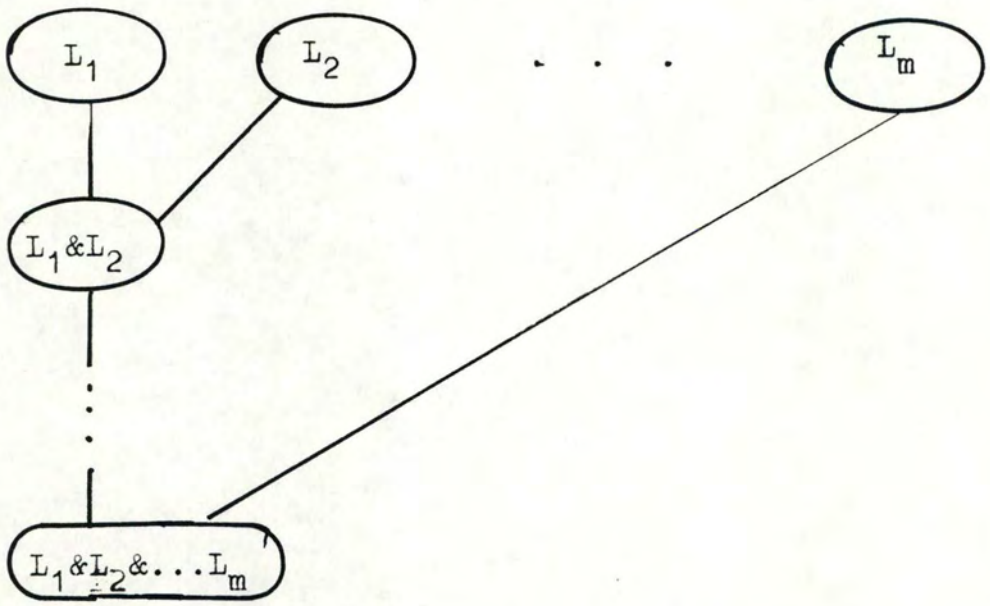


TABLE DES MATIERES

AVANT-PROPOS

TABLE DES MATIERES I

INTRODUCTION 1

CHAPITRE 1 - LES RESEAUX DE PETRI 5

1.1. INTRODUCTION 5
1.2. DEFINITION ET FONCTIONNEMENT DES RESEAUX DE PETRI 6
1.2.1. Définition 1 - Réseau de Petri 6
1.2.2. Définition 2 - Réseau marqué 6
1.2.3. Définition 3 - Transition sensibilisée 6
1.2.4. Définition 4 - Marquage accessible 7
1.2.5. Définition 5 - Graphe des marquages accessibles 8
1.2.6. Définition 6 - Langage d'un réseau de Petri 8
1.2.7. Définition 7 - Place k-bornée 8
1.2.8. Définition 8 - Réseau p-borné 9
1.2.9. Définition 9 - Réseau t-borné 9
1.2.10. Définition 10 - Transition quasi-vivante 9
1.2.11. Définition 11 - Transition vivante 9
1.2.12. Définition 12 - Marquage puits 9
1.2.13. Définition 13 - Réseau réinitialisable 10
1.3. CONCLUSION 11

CHAPITRE 2 - LES RESEAUX DE PETRI TEMPORELS 12

2.1. INTRODUCTION 12

2.2. LES PRINCIPAUX MODELES 13
2.3. LE MODELE DES RESEAUX DE PETRI TEMPORELS 14
2.3.1. Représentation graphique 14
2.3.2. Définition formelle 15
2.3.3. La notion d'état 15
2.3.4. Comportement d'un réseau de Petri temporel 16
A) Conditions de tir d'une transition 16
B) Calcul de l'état suivant 16
C) Relation d'accessibilité - Echéancier de tir 17
2.3.5. Illustration par un exemple 18
2.4. CLASSES D'ETATS ET METHODE PAR ENUMERATION 21
2.4.1. Notion de domaine de tir 21
2.4.2. Définition des classes d'états 21
2.4.3. Calcul des classes d'états 23
A) Règle de tir d'une transition 23
B) Calcul de la classe suivante 23
C) Egalité de deux classes d'états 26
D) Construction du graphe des classes d'états 26
E) Remarques 27
2.4.4. Cas des transitions multi-sensibilisées 27
2.5. CONCLUSION 29

CHAPITRE 3 - PUISSANCE ET PROPRIETES DU MODELE DES RESEAUX DE PETRI TEMPORELS 30

3.1. INTRODUCTION 30
3.2. PUISSANCE D'EXPRESSION DU MODELE DES RESEAUX DE PETRI TEMPORELS 31
3.2.1. Simulation du comportement des réseaux de Petri classiques 31
3.2.2. Simulation des réseaux de Petri temporisés 31
A) Le modèle temporisé de Ramchandani 31
C) Le modèle temporisé de Sifakis 32
C) Le modèle temporisé de Razouk 32
3.2.3. Simulation des réseaux avec arcs inhibiteurs 33
3.3. PROPRIETES SPECIFIQUES 34
3.3.1. Définition 1 - Réseau de Petri temporel bloquant 34
3.3.2. Définition 2 - Réseau de Petri temporel vivant 34
3.3.3. Définition 3 - Réseau de Petri temporel réinitialisable 34
3.3.4. Indécidabilité de certaines propriétés 35

et condition suffisante 39
3.3. CONCLUSION 41

CHAPITRE 4 - LES OUTILS RESEAUX DE PETRI TEMPORELS 42

4.1. INTRODUCTION 42
4.2. CONCEPTION D'UN OUTIL POUR L'ANALYSE DES RESEAUX DE PETRI TEMPORELS 44
4.2.1. Architecture générale 44
4.2.2. L'édition des réseaux de Petri 45
Principes de représentation 45
1° La déclaration des places 45
2° La spécification de l'état initial 46
3° La déclaration des transitions 46
4.2.3. Le simulateur 47
A) Principe d'implémentation 47
B) Fonctionnement du simulateur 47
4.2.4. Le générateur du graphe des classes d'états 48
A) Principes de fonctionnement 48
B) Génération du graphe des classes d'états 49
4.2.5. Analyseur du graphe des classes d'états 50
4.3. RdP : OUTIL D'AIDE A LA CONCEPTION ET A L'EVALUATION DE SYSTEMES PAR RESEAUX DE PETRI 51
4.3.1. Présentation générale 51
4.3.2. Fonctionnalités de RdP 51
A) L'éditeur 51
1° Principaux concepts 51
2° Méthode de description 52
. La vue hiérarchique 52
. La vue communication 52
. La vue réseau de Petri 53
B) L'analyseur 53
1° Différents analyseurs 53
2° L'analyseur de réseaux de Petri temporels 54
. Préparation de la session d'énumération des classes d'états 54
. Lancement de la session d'énumération des classes d'états 54
. Traitement du graphe des classes d'états 55

4.3.3. Critiques et perspectives de travail 55
4.4. CONCLUSION 58

CHAPITRE 5 - METHODOLOGIE D'ANALYSE ET UTILITE DES RESEAUX DE PETRI TEMPORELS 59

5.1. INTRODUCTION 59
5.2. METHODOLOGIE D'ANALYSE 60
5.2.1. Compréhension des spécifications informelles 60
5.2.2. Création du modèle de description formelle sous forme d'un réseau de Petri temporel 60
A) Modèle local d'une entité de protocole 61
B) Modélisation du service utilisé 62
C) Modèle global de la couche de protocole 62
. Communication par places partagées 63
. Communication par rendez-vous 63
5.2.3. Test de fonctionnement du modèle de description formelle 63
5.2.4. Génération du graphe des classes d'états 64
5.2.5. Vérification des propriétés du modèle et analyse de son comportement 64
A) Propriétés générales 64
B) Propriétés spécifiques 65
5.3. APPLICATIONS 66
5.3.1. Le protocole de transfert de données à sens unique 66
A) Présentation 66
B) Modélisation du protocole 66
1° Modèle des réseaux de Petri classiques 66
. Modélisation 66
. Analyse 67
2° Modèle des réseaux de Petri temporels 67
. Modélisation 67
. Analyse 67
5.3.2. Le protocole du bit alterné 69
A) Présentation 69
B) Modélisation du protocole 70
C) Analyse 72
5.3.3. Quelques remarques 74
5.4. CONCLUSION 75

CHAPITRE 6 - SPECIFICATION FORMELLE DU PROTOCOLE DE SCRUTATION CYCLIQUE DU RESEAU FIP **76**

6.1. INTRODUCTION 76
6.2. PRESENTATION DE FIP 77
6.2.1. Le protocole FIP 77
A) Présentation des trois couches 78
1° La couche application 78
a) La promptitude 79
b) Le rafraîchissement 79
c) Service de synchronisation 79
2° La couche liaison de données 80
. Fonctionnement 80
. Service périodique et apériodique 81
. Services offerts par la couche liaison de données à la couche application. 81
3° La couche physique 82
6.3. SPECIFICATION FORMELLE ET ANALYSE DU PROTOCOLE DE SCRUTATION CYCLIQUE DU RESEAU FIP AU MOYEN DES RESEAUX DE PETRI TEMPORELS 83
6.3.1. Hypothèses simplificatrices 83
6.3.2. Les contraintes temporelles 83
A) Présentation et fonctions des contraintes temporelles 83
B) Relations entre les différentes contraintes temporelles 85
C) Interprétation des relations 86
6.3.3. Le modèle des réseaux de Petri temporels 87
A) Extension du modèle des réseaux de Petri temporels 87
B) Découpe en modules 87
1° L'arbitre de bus 87
2° Le producteur 88
3° Le consommateur 89
4° Le médium 89
C) Le modèle global 90
6.3.4. Modélisation à l'aide de l'outil RdP 92
6.3.5. Analyse des temporisations et résultats 94
A) Valeur minimale des bornes inférieures 95
B) Valeur maximale des bornes supérieures 96
C) Relations entre les différentes temporisations 96
6.4. CONCLUSION 98

CONCLUSION **100**

BIBLIOGRAPHIE **103**

ANNEXES ..... **A**

Annexe 1 - Texte source du simulateur A-1
Annexe 2 - Modèle fip_per : architecture A-2
Annexe 3 - Modèle fip_per : résultats de l'analyse (fonctionnement du système sans erreur) A-3
Annexe 4 - Modèle fip_per : résultats de l'analyse (fonctionnement du système avec erreurs) A-4

INTRODUCTION

Une évolution évidente de la structure des systèmes informatiques est aujourd'hui le développement de systèmes répartis, composés de nombreuses stations, serveurs ou automates.

L'interconnexion de ces équipements informatiques conduit à concevoir des réseaux de communication complexes, nécessitant la mise en oeuvre de tout un ensemble de règles gérant les échanges d'information entre les divers correspondants. Ces règles définissent les protocoles de communication.

La complexité de ces protocoles nécessite d'aborder leur conception à partir d'une méthodologie générale [AYA 85a], qui comprend quatre étapes fondamentales :

1. Définition du cahier des charges, destinée à établir de manière informelle et aussi claire que possible les spécifications fonctionnelles du système.
2. Formulation de ces spécifications avec un modèle de description formelle et vérification à l'aide de techniques de validation associée au modèle formel considéré.
3. Implémentation du système suivant des techniques qui découlent du modèle formel.
4. Tests de certification de bon fonctionnement.

L'ensemble des résultats obtenus lors de la validation doit pouvoir permettre l'acceptation ou le rejet du cahier des charges, c'est-à-dire des spécifications.

Il importe donc que le modèle formel choisi pour la vérification soit bien adapté au problème à traiter afin que les résulats obtenus soient sûrs.

Le choix du modèle formel est guidé par les caractéristiques principales du système à modéliser.

Dans le cas des protocoles de communication, il doit permettre d'exprimer les concepts fondamentaux des systèmes distribués à savoir, le parallélisme, la concurrence et la synchronisation.

Parmi tous les travaux menés tant au niveau théorique qu'au niveau applications, il apparaît que les réseaux de Petri et les modèles qui en sont dérivés, sont tout à fait adaptés à la description des systèmes distribués. En effet, des résultats intéressants ont été obtenus et des applications significatives ont été menées. Toutefois, si les réseaux de Petri permettent d'exprimer des contraintes de séquencement, c'est-à-dire des actions liées à un temps logique, ils ne permettent pas de représenter de manière explicite le temps réel. Or, il est souvent nécessaire de pouvoir manipuler des temporisations afin de représenter les durées d'exécution ou des temps d'attente. C'est dans ce but que les réseaux de Petri temporisés ont été définis. Ils permettent de représenter des systèmes avec temporisations, ce qui est particulièrement important pour les protocoles de communication, comme par exemple dans le cas de systèmes tolérants aux pannes. Si l'on considère en effet de tels systèmes, nous constatons que dans une grande proportion, leur fonctionnement est basé sur des dispositifs de sécurité appelés temps-limites et dont l'action est exclusivement dépendante du temps.

Parmi les modèles des réseaux de Petri prenant en compte le temps, nous en retiendrons principalement deux : d'une part, le modèle de Ramchandani [RAM 74] qui associe à chaque transition du réseau de Petri, un paramètre temporel correspondant à la durée de tir de la transition; et, d'autre part, le modèle de Merlin [MER 74] qui associe à chaque transition du réseau de Petri deux paramètres temporels, le premier correspondant au délai minimum qu'une transition sensibilisée doit attendre avant de pouvoir être tirée et le second au délai maximum pendant lequel elle peut rester sensibilisée sans être tirée.

Dans ce mémoire, nous avons choisi d'analyser le modèle de Merlin et la méthode d'analyse qui y est associée. Ce modèle s'est avéré adéquat [BER 83b] pour exprimer la plupart des contraintes temporelles des systèmes distribués alors que de telles contraintes étaient difficiles à exprimer en terme de durée de tir.

Au cours de ce travail, nous nous attacherons à montrer la puissance du modèle de Merlin, communément appelé le modèle des réseaux de Petri temporels, et des techniques de vérification sous-jacentes.

Dans le premier chapitre, nous rappelons les propriétés essentielles des réseaux de Petri classiques auxquelles nous avons eu recours pour l'analyse du modèle temporel.

Le deuxième chapitre est consacré à l'analyse des réseaux de Petri temporels. Nous y présentons une méthode d'analyse développée au Laboratoire d'Automatique et d'Analyse des Systèmes de Toulouse.

Cette méthode repose sur l'énumération de classes d'états et sur la construction d'un graphe permettant une analyse d'accessibilité semblable à la méthode utilisée pour l'analyse des réseaux de Petri classiques.

Il faut noter que si le réseau est non borné, ce graphe est également non fini et il devient dès lors impossible de mener à bien une analyse. Nous montrerons que le caractère borné pour un réseau de Petri temporel n'est pas décidable. C'est pour cette raison, que des conditions suffisantes pour qu'un réseau de Petri temporel soit borné ont été recherchées [MEN 82]. Ces conditions suffisantes, mais non nécessaires, permettent en fait de distinguer deux classes de réseaux : ceux dont on peut dire qu'ils sont bornés, et ceux pour lesquels on ne peut décider s'ils sont bornés ou non.

Au chapitre 3, nous présentons et explicitons l'une de ces conditions suffisantes . Nous avons choisi de présenter celle-là et pas une autre, parce qu'elle est apparue la plus satisfaisante dans la majorité des exemples traités [MEN 82]. De plus, nous l'avons retrouvée dans la plupart des outils réseaux de Petri temporels que nous avons rencontrés.

Dans ce même chapitre, nous redéfinissons des propriétés de réseaux de Petri temporels similaires à celles des réseaux de Petri classiques telles que les propriétés bloquant, vivant, réinitialisable.

De même, nous montrons la puissance du modèle de Merlin, en donnant des relations existant entre les diverses classes de réseaux de Petri (par exemple, les différents modèles temporisés, les réseaux de Petri avec arcs inhibiteurs, ...) et les réseaux de Petri temporels.

Lorsque les systèmes deviennent complexes, ce qui est souvent le cas pour les protocoles de communication, l'étude de ces systèmes nécessite le recours à l'utilisation d'outils logiciels.

C'est pourquoi, nous donnons au chapitre 4, une approche pour la conception de tels outils, approche que nous avons d'ailleurs en partie concrétisée par la conception d'un petit simulateur de réseaux de Petri temporels.

La suite de ce chapitre est consacrée à la présentation d'un outil professionnel d'aide à la conception et à l'évaluation de systèmes par réseaux de Petri. Cet outil, dénommé RdP, a été réalisé par la société Vérilog de Toulouse. Il facilite la modélisation et l'analyse des systèmes importants en permettant l'utilisation de sous modèles qui sont ensuite connectés entre eux automatiquement.

Il faut souligner que l'important dans de tels logiciels, c'est qu'ils doivent permettre d'utiliser les techniques de vérification sous-jacentes au modèle utilisé et qu'ils doivent s'insérer dans ou permettre d'utiliser une approche méthodologique de conception de systèmes [AYA 85a].

Une telle approche est développée au chapitre 5 et montre clairement la nécessité de recourir à ces outils.

Egalement au chapitre 5 et afin d'illustrer les points précédents, nous présentons deux exemples : l'un concerne la modélisation du protocole de transfert de données à sens unique et l'autre, la modélisation du protocole du bit alterné. Ces deux exemples vont nous permettre de montrer l'utilité du modèle de Merlin et ses avantages par rapport aux réseaux classiques.

Finalement, le sixième chapitre est consacré à une application significative du modèle des réseaux de Petri temporels à la modélisation et à la vérification d'un protocole de communication. Le protocole considéré est celui de la couche liaison de données du réseau local industriel FIP. Nous étudions les contraintes temporelles de ce protocole et utilisons à cet effet l'outil RdP.

CHAPITRE 1

LES RESEAUX DE PETRI

1.1. Introduction

Avant d'aborder le formalisme des réseaux de Petri temporels, nous allons rappeler dans ce chapitre quelques définitions relatives aux réseaux de Petri classiques qui nous aideront à mieux comprendre la suite de ce mémoire.

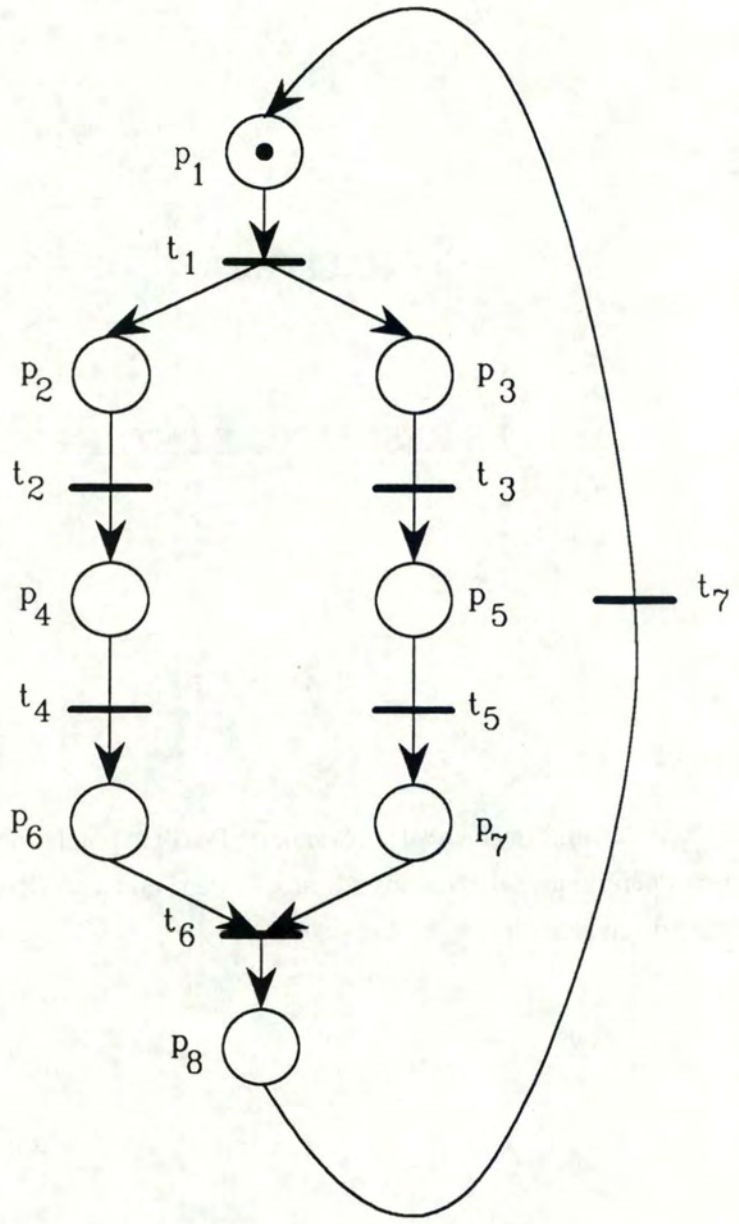


Figure 1.1. Réseau de Petri classique

1.2. Définition et fonctionnement des réseaux de Petri

[BRA 83a] [FIC 88] [PET 81]

1.2.1. Définition 1

Un *réseau de Petri* est un quadruplet $R = <P,T;Pré,Post>$ où :

. P est un ensemble fini de places pour lequel on note $m = |P|$

. T est un ensemble fini de transitions, disjoint de P pour lequel on note $n = |T|$

. Pré : $P \times T \rightarrow N : (p,t) \rightarrow Pré(p,t)$ est l'application d'incidence avant

. Post : $P \times T \rightarrow N : (p,t) \rightarrow Post(p,t)$ est l'application d'incidence arrière

Représentation graphique

Un réseau de Petri peut être représenté graphiquement (Cf. figure 1.1.) par un graphe orienté où les places représentées par des cercles et les transitions représentées par des rectangles sont les noeuds de ce graphe. Les arcs relient une place p à une transition t (respectivement une transition t à une place p) si, et seulement si, $Pré(p,t) \neq 0$ (respectivement $Post(p,t) \neq 0$).

La valeur Pré(p,t) est appelée le "poids" de l'arc qui relie la place p à la transition t et désigne le nombre de ressources Pré(p,t) dont la transition t a besoin sur la place p. De même, la valeur Post(p,t) est appelée le "poids" de l'arc qui relie la transition t à la place p et donne le nombre de ressources Post(p,t) que la transition t crée dans la place p.

1.2.2. Définition 2

Un *réseau marqué* est le couple N = (R;M) formé d'un réseau R et d'une application $M : P \rightarrow N$ appelée marquage.

M(p) est le marquage de la place p, on dit aussi le nombre de marques contenues dans p ou encore le "contenu en jetons" de la place p.

Un marquage spécial, M_0, représente le marquage initial du réseau.

1.2.3. Définition 3

Dans un réseau marqué N = (R;M), avec R = (P,T;Pré,Post), une *transition* t est dite *sensibilisée* par le marquage M ou encore *franchissable* pour le marquage M si, et seulement si :

$$\forall p \in P, M(p) \geq Pré(p,t)$$

On note M(t> cette relation dans N^m X T : c'est la *précondition de franchissement* de t.

De même, pour k ≥ 1, une *transition* t est dite *k-sensibilisée* par un marquage M si, et seulement si :

$$\forall p \in P, M(p) \geq k \,.\, Pré(p,t)$$

Quel que soit k ≥ 1, toute transition k-sensibilisée par M est dite simplement sensibilisée par M.

La précondition de franchissement étant vérifiée, la transition t peut être tirée. Le franchissement de t permet d'obtenir le nouveau marquage M' tel que :

$$\forall p \in P, M'(p) = M(p) - Pré(p,t) + Post(p,t)$$

ou encore

$$M' = M - Pré(.,t) + Post(.,t)$$

où

Pré(.,t) est le marquage minimum permettant le franchissement de la transition t,

Post(.t) est le marquage minimum que l'on puisse obtenir par un franchissement de t.

On note M(t>M' la relation correspondante dans N^m X T X N^m pour indiquer que M(t> et que le résultat du tir de la transition t est le marquage M'.

Etant donné la précondition M ≥ Pré(.,t), le franchissement de t entraîne M' ≥ Post(.,t). C'est la postcondition pour qu'un marquage M' puisse résulter du franchissement de la transition t.

Les règles d'évolution ci-dessus définissent une *relation d'accessibilité* sur l'ensemble des marquages d'un réseau de Petri.

1.2.4. Définition 4

Notation

En considérant T^* comme un vocabulaire composé des symboles t des transitions du réseau R, un mot s ∈ T^* dénote une séquence de transitions, le mot vide λ dénotant une séquence vide.

Un *marquage* $\overline{M}$ est *accessible* à partir du marquage M si, et seulement si, il existe une séquence s de transitions successivement tirables, qui conduit de M à $\overline{M}$.

L'*ensemble des marquages accessibles* à partir de M dans le réseau R est :

$$A(R;M) = \{\overline{M} \in N^m \mid \exists\, s : M(s>\overline{M}\}$$

On note :

$M(*>\overline{M}$ si, et seulement si, $\overline{M} \in A(R;M)$

$M(+>\overline{M})$ si, et seulement si, $\exists\, s \neq \lambda : M(s>\overline{M}$

1.2.5. Définition 5

On appelle *graphe des marquages accessibles* GA(R;M), à partir du marquage M dans le réseau R, le graphe :

- ayant A(R,M) pour ensemble de sommets,
- dans lequel (M',M'') est un arc joignant le marquage M' au marquage M'' si, et seulement si :

$M(*>M'$ et $\exists\, t \in T : M'(t>M''$

Le graphe des marquages représente un diagramme d'états du comportement dynamique du réseau de Petri correspondant où chaque état est représenté par un marquage accessible.

1.2.6. Définition 6

L'ensemble de toutes les séquences de transitions tirables d'un réseau marqué N = (R;M) forme le *langage* :

$$L(R;M) = \{s \in T^* \mid M(s>\}$$

Si l'on étiquette chaque arc du graphe des marquages accessibles par la transition correspondante, l'ensemble des mots formés en concaténant les étiquettes des arcs constituant chacun des chemins du graphe des marquages accessibles GA(R;M) forme le langage L(R;M).

1.2.7. Définition 7

Une *place* p d'un réseau marqué $N = (R;M_0)$ est dite *k-bornée* ($k \in N$) si pour tout marquage accessible $M \in A(R;M_0) : M(p) \leq k$.

Dans le cas contraire, la place p est dite non bornée.

Si *p* est *1-bornée*, elle est appelée *place binaire*.

1.2.8. Définition 8

Un *réseau* marqué $N = (R;M_0)$ est dit *p-borné* ou simplement *borné* si, et seulement si :

$\exists\, k \in N$ tel que $\forall\, M \in A(R;M_0)$ et $\forall\, p \in P$, $M(p) \leq k$

Si $k = 1$, le *réseau* est dit *sauf*. Si $k > 1$, le *réseau* est dit *k-p-borné* ou tout simplement *k-borné*.

1.2.9. Définition 9

Un *réseau* marqué $N = (R;M_0)$ est dit *t-borné* si, et seulement si :

$\exists\, k \in N$ tel que $\forall\, M \in A(R,M_0)$ et $\forall\, t \in T$, $\exists\, p \in P$ tel que

$M(p) < k \,.\, Pré(p,t)$

Si $k = 2$, le *réseau* est dit *t-sauf*. Si $k > 2$, le *réseau* est dit *k-t-borné*.

1.2.10. Définition 10

Une *transition* t d'un réseau marqué $N = (R;M_0)$ est dite *quasi-vivante* si il existe un marquage accessible $M \in A(R;M_0)$ tel que $M(t>$.

Quand toutes les transitions d'un réseau marqué sont quasi-vivantes, on dit que ce *réseau* est *quasi-vivant*.

1.2.11. Définition 11

Une *transition* t d'un réseau marqué $(R;M_0)$ est *vivante* si, pour tout marquage accessible $M \in A(R;M_0)$, t est quasi-vivante pour le réseau (R,M).

Un *réseau marqué* $N = (R;M_0)$ est *vivant* si toutes ses transitions sont vivantes.

1.2.12. Définition 12

Un marquage M d'un réseau marqué $N = (R;M_0)$ est appelé *marquage puits* si aucune transition n'est franchissable depuis M.

Dès lors, si tout marquage accessible depuis M_0 n'est *pas un marquage puits*, le réseau $(R;M_0)$ est dit *sans blocage*.

1.2.13. Définition 13

Un *réseau marqué* $N = (R;M_0)$ est *réinitialisable* si, et seulement si, son graphe des marquages accessibles $GA(R;M_0)$ est fortement connexe :

$$\forall M \in A(R;M_0), \exists s \in T^* : M(s > M_0$$

1.3. Conclusion

Nous venons de rappeler quelques définitions essentielles des réseaux de Petri.

Dans la littérature, de nombreux auteurs ont déjà mis en évidence les avantages que les réseaux de Petri classiques peuvent apporter dans la modélisation, la validation, voire l'implantation de systèmes réels ayant un comportement parallèle et présentant des contraintes de synchronisation.

Parmi les limitations que l'on peut leur attribuer (difficultés pour la représentation de problèmes de grande taille, pour la représentation du traitement des données, ...), il en est une qui nous semble importante et qui est l'absence du temps dans le modèle. Or, de manière générale, les systèmes distribués sont soumis à des contraintes temporelles et leur modélisation nécessite donc le recours à des modèles incluant explicitement le temps. C'est pourquoi, la suite de ce travail sera consacrée à la présentation d'un modèle incluant le paramètre temps.

Le modèle que nous avons retenu est le modèle des réseaux de Petri temporels de Merlin.

Les raisons de ce choix ainsi qu'une méthode d'analyse sont exposées au chapitre suivant.

CHAPITRE 2

LES RESEAUX DE PETRI TEMPORELS

2.1. Introduction

Dans ce chapitre, après avoir situé le modèle des réseaux de Petri temporels, nous donnons le modèle formel et ses règles de fonctionnement associées. Ensuite, nous illustrons cette démarche par un exemple. Cet exemple sera également l'occasion de montrer la difficulté de représenter le fonctionnement d'un réseau temporel par l'ensemble de ses états qui peuvent être en nombre infini. Ceci va nous conduire à considérer une méthode permettant de grouper les états en vue d'obtenir un graphe beaucoup plus facile à manipuler et à analyser; il s'agit de la méthode d'analyse par énumération des classes d'états.

2.2. Les principaux modèles

Généralement, dans la spécification des systèmes on retrouve deux types de temps : le *temps logique* et le *temps physique*. Le *temps logique* est utilisé pour exprimer un ordre entre les événements d'un système ou encore une relation d'accessibilité entre les différents états du système. C'est le cas des réseaux de Petri classiques, des modèles basés sur des systèmes de transitions et aussi de la logique temporelle. On différencie le *temps physique* du *temps logique* par le fait que ce premier peut être quantifié.

Plusieurs tentatives ont été effectuées dans le passé pour analyser le comportement des systèmes parallèles dans lesquels le temps apparaît comme un paramètre quantifiable et continu.

Beaucoup de modèles pour la représentation de tels systèmes ont été dérivés des réseaux de Petri, probablement à cause de leur capacité à représenter des comportements parallèles et/ou asynchrones. Parmi ces modèles, nous retiendrons plus particulièrement ceux développés parallèlement mais indépendamment l'un de l'autre en 1974, par Ramchandani au MIT [RAM 74] et par Merlin à l'université de Californie [MER 74].

Le modèle développé par Ramchandani, que nous appellerons réseaux de Petri temporisés, associe à chaque transition d'un réseau de Petri classique un seul paramètre temporel correspondant à la durée de tir de cette transition.

Dès lors, au contraire des réseaux de Petri classiques, le tir d'une transition peut être non instantané.

Ces réseaux ont été principalement utilisés pour l'analyse de performances.

Un autre modèle temporisé a été introduit plus tard, en 1977, par Sifakis [SIF 77]; ce modèle associe à chaque place du réseau un paramètre temporel dont la sémantique correspond au temps pendant lequel un jeton déposé dans une place doit rester indisponible; seul les jetons disponibles sont considérés pour tester si une transition peut être tirée.

Sifakis a montré, en 1979, que son modèle était équivalent à celui de Ramchandani [SIF 79]. Toutefois, l'avantage du modèle de Sifakis est de préserver la caractéristique de "tir instantané" des transitions dans un réseau de Petri.

Sifakis a également orienté ses travaux vers l'analyse de performances tels que les temps de réponse et les débits des systèmes.

Quant au modèle de Merlin, qui nous intéresse plus paticulièrement et que nous désignerons sous le terme de réseaux de Petri temporels, il associe à chaque transition du réseau de Petri deux paramètres temporels a et b, où $0 \leq a \leq b$ avec b pouvant être non borné. Le paramètre a correspond au délai minimum qu'une transition sensibilisée doit attendre avant de pouvoir être tirée et b au délai maximum pendant lequel elle peut être sensibilisée sans être tirée. Ces temps a et b sont relatifs à la date à laquelle la transition, soit t, a été sensibilisée pour la dernière fois. Ainsi, si t est sensibilisée au temps θ et n'est pas désensibilisée par le tir d'une

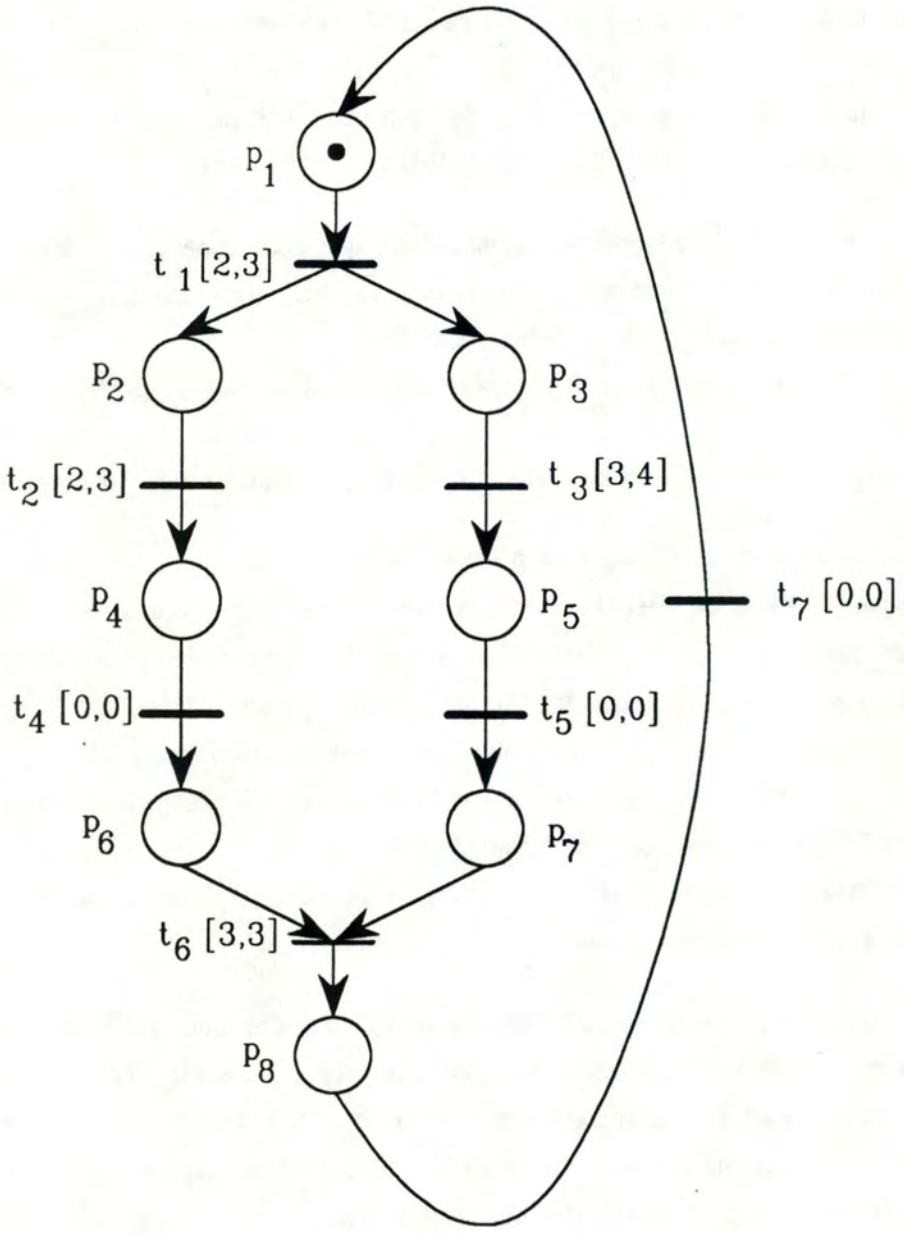


Figure 2.1. Réseau de Petri temporel

autre transition, t ne peut être tirée avant le temps θ+a et doit l'être au plus tard au temps θ+b. Notons encore que le tir des transitions est ici de durée nulle.

Alors qu'il est délicat d'exprimer certaines contraintes seulement à l'aide de durées, le modèle de Merlin permet d'exprimer simplement la plupart des contraintes temporelles, y compris les durées.

L'utilisation du modèle de Merlin semble donc particulièrement bien adaptée pour la modélisation des systèmes soumis à des contraintes temporelles et plus particulièrement pour la conception de protocoles où l'on doit souvent tenir compte d'éléments tels que le temps de réponse, le temps de reconfiguration, ...

De plus, le fait que la plupart des extensions des réseaux de Petri (réseaux avec arcs inhibiteurs, réseaux temporisés, ...) peuvent être simulés, ainsi que nous le montrerons, par un réseau temporel équivalent, renforce cette position.

C'est pourquoi, l'équipe Logiciel et Communication du Laboratoire d'Automatique et d'Analyse des Systèmes à Toulouse a opté pour le choix du modèle de Merlin et a orienté une partie de ses travaux dans cette direction.

2.3. Le modèle des réseaux de Petri temporels

[BER 82] [BER 83b] [MEN 82] [MER 74] [ROU 85b]

2.3.1. Représentation graphique

Ainsi que nous venons de le dire, les réseaux de Petri temporels sont obtenus à partir des réseaux de Petri classiques en associant à chaque transition un intervalle fermé de temps. La borne inférieure de cet intervalle, pour une transition t, correspond à la *Date Statique de tir au plus tôt* de t, soit DSmin(t), et la borne supérieure correspond à la *Date Statique de tir au plus tard* de t, soit DSMax(t). L'intervalle formé à partir des dates statiques de tir au plus tôt et au plus tard est appelé *Intervalle Statique de tir*.

A la figure 2.1., on trouve un exemple de réseau de Petri temporel. Cet exemple sera utilisé tout au long de ce chapitre pour illustrer les concepts et méthodes introduites. Notons qu'aucune signification particulière n'est attachée à ce réseau.

2.3.2. Définition formelle

Un *réseau de Petri temporel*, noté R_Θ, est défini par le triplet suivant :

$$R_\Theta = (R;M_0,I_0)$$

où R = (P,T;Pré,Post) est un réseau de Petri,

$M_0 : P \rightarrow N$ est le marquage initial,

$I_0 : T \rightarrow (Q^+ \cup 0) \times (Q^+ \cup 0 \cup \infty)$
est l'*application intervalle de temps initial*

$t_i \rightarrow I_0(t_i) = [DSmin(t_i),DSMax(t_i)]$
avec $0 \leq DSmin(t_i) \leq DSMax(t_i)$ et $\forall$ i, $1 \leq i \leq n$ avec $n = |T|$

L'*application intervalle de temps* définit un intervalle de dates de tir autorisées pour les transitions. Il faut encore noter que l'intervalle n'est validé qu'à partir du moment où la transition t_i est sensibilisée.

Notons également que lorsque le réseau évolue, les intervalles de tir des transitions peuvent également évoluer et ainsi différer des intervalles statiques. C'est pourquoi, nous désignerons les bornes inférieure et supérieure de l'intervalle de tir $I(t_i)$, pour une transition t_i, simplement par a_i et b_i, représentant respectivement la *date de tir au plus tôt* et la *date de tir au plus tard* de la transition t_i.

2.3.3. La notion d'état

Avant d'exposer les règles de fonctionnement d'un réseau de Petri temporel, il nous faut définir la notion d'état.

Un *état* dans un réseau de Petri temporel R_Θ est un couple E = (M,I) pour lequel :

. M est un marquage de R_Θ tel que $\forall p \in P$, $M(p) \geq 0$

. I est l'application intervalle de tir qui fait correspondre à chaque transition l'intervalle de temps dans lequel elle peut être tirée.

On aura :

$\forall t_i \in T$, avec $1 \leq i \leq n$ et $n = |T|$: $I(t_i) = \varnothing$

ssi

t_i non sensibilisée

2.3.4. Comportement d'un réseau de Petri temporel

A) Conditions de tir d'une transition

Le calcul de l'ensemble des états d'un réseau de Petri temporel et la relation d'accessibilité sur cet ensemble sont basés sur la règle de tir d'une transition.

Supposons que l'état courant du réseau de Petri temporel est E = (M,I). Certaines transitions peuvent être sensibilisées par le marquage M, mais parmi celles-ci, toutes ne peuvent être tirées depuis l'état E, cela en raison des intervalles de temps associés aux transitions.

Ceci nous amène à définir les conditions de tir d'une transition.

Dans un réseau réseau de Petri temporel R_Θ, une transition t est tirable à un temps θ, depuis l'état E = (M,I), si, et seulement si, les deux conditions suivantes sont satisfaites :

1. La transition est sensibilisée par le marquage M.

2. θ est compris entre la date de tir au plus tôt de la transition t et la plus petite des dates de tir au plus tard des transitions sensibilisées par le marquage.
   Notons également que l'origine du temps pour θ est l'instant où l'état E a été atteint.

La première condition est indépendante du temps θ. La seconde condition distingue une transition tirable dans un réseau de Petri temporel d'une transition tirable dans un réseau de Petri classique. Elle traduit le fait qu'une transition sensibilisée ne peut être tirée avant son instant de tir au plus tôt et doit être tirée avant ou pendant son instant de tir au plus tard à moins que le tir d'une autre transition se produise avant et la désensibilise.

La deuxième condition souligne également le fait que l'intervalle de temps pendant lequel une transition peut être tirée tant que le réseau se trouve dans un état donné n'est pas forcément le même que celui admissible pour cette transition à partir de cet état.

B) Calcul de l'état suivant

Le tir d'une transition tirable, soit t, depuis un état E = (M,D), à un instant θ, conduit à un nouvel état E' = (M',I') où :

1. Le nouveau marquage M' est défini comme pour les réseaux de Petri classiques :
   $\forall p \in P : M'(p) = M(p) - Pré(p,t) + Post(p,t)$

2. Les nouveaux intervalles de tir pour les transitions sont obtenus comme suit :
   a) Pour toutes les transitions t_i non sensibilisées par le nouveau marquage M', $I'(t_i)$ est vide.

b) Pour toutes les transitions t_j qui étaient sensibilisées par le marquage M et non en conflit avec la transition t, c'est-à-dire les transitions qui sont restées sensibilisées,

$$I'(t_j) = [\max(0,a_j-\theta),b_j-\theta] = [a_j',b_j']$$

où a_j et b_j désignent respectivement les dates de tir au plus tôt et au plus tard de la transition t_j. On prend le maximun de $(0,a_j-\theta)$ parce que le temps est positif.

c) Toutes les autres transitions reçoivent leurs intervalles de tir initiaux, c'est-à-dire les intervalles de tir statiques associés à ces transitions.

En d'autres mots, suite au tir de la transition t, les transitions non sensibilisées par le nouveau marquage M' reçoivent des intervalles de tir vides. Les transitions sensibilisées par M et non en conflit avec t, c'est-à-dire les transitions restées sensibilisées pendant le tir de t, ont leur intervalle de tir décalé de la valeur θ vers l'origine du temps et restreint si nécessaire aux seules valeurs positives. Rappelons ici que l'instant θ est relatif au moment où l'état E(M,I) a été atteint. Les transitions restantes, c'est-à-dire celles nouvellement sensibilisées par M' et celles sensibilisées par M' mais qui étaient déjà sensibilisées par M et en conflit avec t dans M, reçoivent, pour intervalle, leur intervalle statique.

C) Relation d'accessibilité - Echéancier de tir

La règle de tir présentée ci-dessus définit une relation d'accessibilité sur les états du réseau de Petri temporel. Les séquences de tir sont définies dans les réseaux de Petri temporels de la même manière que pour les réseaux de Petri classiques. Dès lors, on peut définir un échéancier de tir dans un réseau de Petri temporel comme une paire constituée d'une séquence de tir, soit s, et d'une séquence de temps correspondant un à un aux transitions de s, soit u.

$$(s,u) = (t_1,\theta_1) . (t_2,\theta_2) . \ldots\ldots . (t_n,\theta_n)$$

où $t_1, t_2, \ldots, t_n$ sont des transitions et

$\theta_1, \theta_2, \ldots, \theta_1$ des temps

L'échéancier (s,u) est réalisable depuis un état donné E_0 ssi les transitions dans s sont successivement tirables depuis cet état, aux temps relatifs leur correspondant dans u.

$$E_0 \xrightarrow{(t_1,\theta_1)} E_1 \xrightarrow{(t_2,\theta_2)} E_2 \longrightarrow \ldots \longrightarrow E_{n-1} \xrightarrow{(t_n,\theta_n)} E_n$$

Dès lors, le comportement d'un réseau de Petri temporel, peut être non seulement caractérisé par l'ensemble de ses états accessibles, mais aussi par l'ensemble de ses échéanciers réalisables depuis l'état initial.

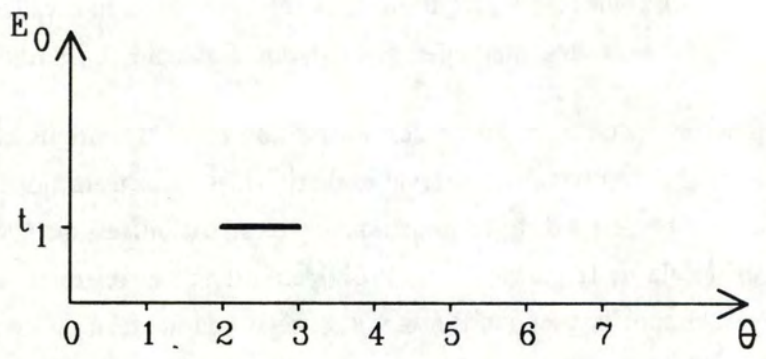


Figure 2.2.

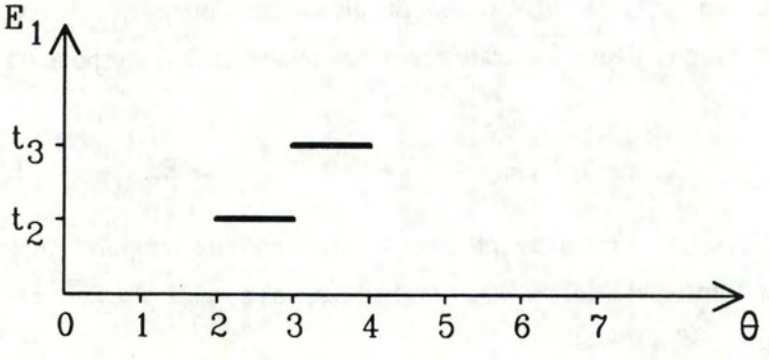


Figure 2.3.

2.3.5. Illustration par un exemple

A présent, illustrons les conditions de tir et les règles d'évolution sur le réseau donné en exemple à la figure 2.1. et dont le marquage initial est représenté par le mot p_1.

Au marquage initial correspondant au marquage $M_0 = p_1$ et au temps 0, seule la transition t_1 est sensibilisée et peut être tirée dans l'intervalle de temps fermé [2,3], compté à partir du moment où la transition a été sensibilisée.

Notons que dans l'état initial, l'instant de sensibilisation des transitions se confond toujours avec l'instant où l'état a été atteint, soit l'instant 0.

L'état inital $E_0 = (M_0, I_0)$ est défini par

$M_0 : p_1(1)$

$I_0 : 2 \leq \underline{t}_1 \leq 3$

où

. $p_1(1)$ signifie que la place p_1 est marquée par un jeton.

. $\underline{t}_1$ est la variable instant de tir associée à la transition t_1; une autre notation est donnée par $\theta(t_1)$.

Dans le cas présent, la transition t_1 peut être tirée à partir de E_0 dans l'intervalle [2,3]. Une illustration graphique de cela est donnée par la figure 2.2.

Supposons que nous tirons t_1 à un temps θ_1 permis. Nous accédons alors à un nouvel état $E_1(M_1, I_1)$ tel que :

$M_1 : p_2(1), p_3(1)$

$I_1 : 2 \leq \underline{t}_2 \leq 3$

$\quad 3 \leq \underline{t}_3 \leq 4$

Quel que soit l'instant exact de tir de t_1 entre les instants 2 et 3, on obtiendra le même état puisque les intervalles de temps sont toujours définis à partir de l'instant de sensibilisation des transitions. Or, dans E_1, les transitions t_2 et t_3 ont été sensibilisées après le tir de t_1. Le temps θ_1 n'apparaît donc pas dans ce système.

Cette notation explicite de I sous la forme d'un système d'inéquations montre quelles sont les contraintes imposées sur les dates de tir des transitions sensibilisées. On peut dire que ce système illustre les plans de tir pour le futur.

Tous les plans de tir futurs peuvent être représentés sous la forme d'un graphique et chacun des plans qui peut être représenté par l'intermédiaire d'un vecteur $\underline{t} = (\underline{t}_2, \underline{t}_3)$ tel que $\underline{t}_2 \in [2,3]$ et $\underline{t}_3 \in [3,4]$ est une possibilité d'évolution (Cf. figure 2.3.).

Par exemple, $\underline{t}_2 = 2{,}4$ et $\underline{t}_3 = 3$ est une perspective de tir pour le futur à partir de E_1.

D'après les conditions de tir d'une transition, t_2 et t_3 peuvent être tirées.

La transition t_2 peut être tirée à tout instant compris dans l'intervalle [2,3] tandis que t_3 peut être seulement tirée dans l'intervalle [3,3] car la borne supérieure de l'intervalle de tir de t_3 doit être égale au plus petit des instant de tir au plus tard des transitions sensibilisées. Ceci s'explique par le fait qu'à l'instant strictement supérieur à 3, l'état présent sera sûrement autre que E1 puisque t_2 doit être tirée entre 2 et 3.

. *Considérons le tir de t_3*

Le seul intervalle commun où les transitions sensibilisées peuvent être tirées est l'intervalle [3,3].

Si t_3 est tirée, ce ne peut être qu'à l'instant 3 et bien que les tirs des transitions soient indépendants, cela implique que t_2 soit également tirée à l'instant 3.

Pour s'en sortir, il convient de faire la considération suivante : *la simultanéité parfaite n'existe pas* [MEN 82].

Ainsi, nous pouvons dire que deux événements indépendants, dont l'occurrence est presque simultanée, peuvent être observés avec un ordre différent. Dès lors, à l'instant 3, les deux séquences de tir, t_2t_3 ou t_3t_2, peuvent être présentes indistinctement. Pour un observateur incapable de distinguer les instants de tir de t_2 et de t_3, il peut les considérer comme simultanés [MEN 82].

Le nouvel état engendré par le tir de la transition t_3 à l'instant 3 est donné par $E_2 = (M_2,I_2)$ tel que :

$$M_2 : p_2(1),\ p_5(1)$$
$$I_2 : 0 \le t_2 \le 0$$
$$\quad\ 0 \le t_5 \le 0$$

Après le tir de t_3 dans l'intervalle [3,3], il reste pour t_2 l'intervalle [0,0], ce qui confirme le fait que t_2 doit être tirée simultanément avec t_3.

. *Poursuivons à partir de E1 avec le tir de t2 à un instant compris entre 2 et 3*

Soit θ_2 cet instant; si nous prenons comme nouvelle origine de temps l'instant du dernier tir de transition, en l'occurrence le tir de t_1 depuis l'état E_0, quel que soit l'instant θ_2, il restera pour t_3 un intervalle de temps égal à $[3-\theta_2, 4-\theta_2]$, avec θ_2 variant entre 2 et 3; l'état suivant sera défini par $E_3 = (M_3,I_3)$:

$$M_3 : p_3(1),\ p_4(1)$$
$$I_3 : \max(0,3-\theta_2) \le t_3 \le 4-\theta_2$$
$$\quad\ 0 \le t_4 \le 0$$

Contrairement à l'état E_1 qui ne gardait pas trace de θ_1, l'état E_3, représenté ci-dessus, concerve à travers θ_2 la trace de tir de la transition t_2 depuis l'état E_1.

En raison du caractère continu du temps et du fait que les transitions peuvent être tirées à tout instant dans leur intervalle de tir, les états définis peuvent avoir une infinité de suivants et l'existence d'une infinité d'états devient dès lors possible. Ainsi, θ_2 peut prendre toute valeur réelle dans l'intervalle [2,3] et le seul tir de t_2 à partir de l'état E_1 va générer un nombre infini d'états accessibles ayant tous le même marquage.

Représenter le fonctionnement d'un réseau de Petri temporel par une énumération exhaustive de ses états, tout comme on représente le fonctionnement d'un réseau de Petri par l'ensemble de ses marquages, est donc, dans le cas général, impossible.

Cependant, dans l'analyse actuelle, ce qui nous intéresse, c'est de pouvoir caractériser le comportement du système, comportement qui est défini par la succession de ses événements internes. Sachant que ces événements sont représentés par les séquences de tir des transitions, ce qui importe, ce n'est pas les instants précis des tirs mais bien les séquences de tir possibles.

Suite à ces considérations, il serait intéressant de pouvoir rassembler cette infinité d'états accessibles en un *état d'exécution* représentatif du comportement du système.

Nous allons voir comment le *concept de classes d'états* permet une représentation finie de ces états.

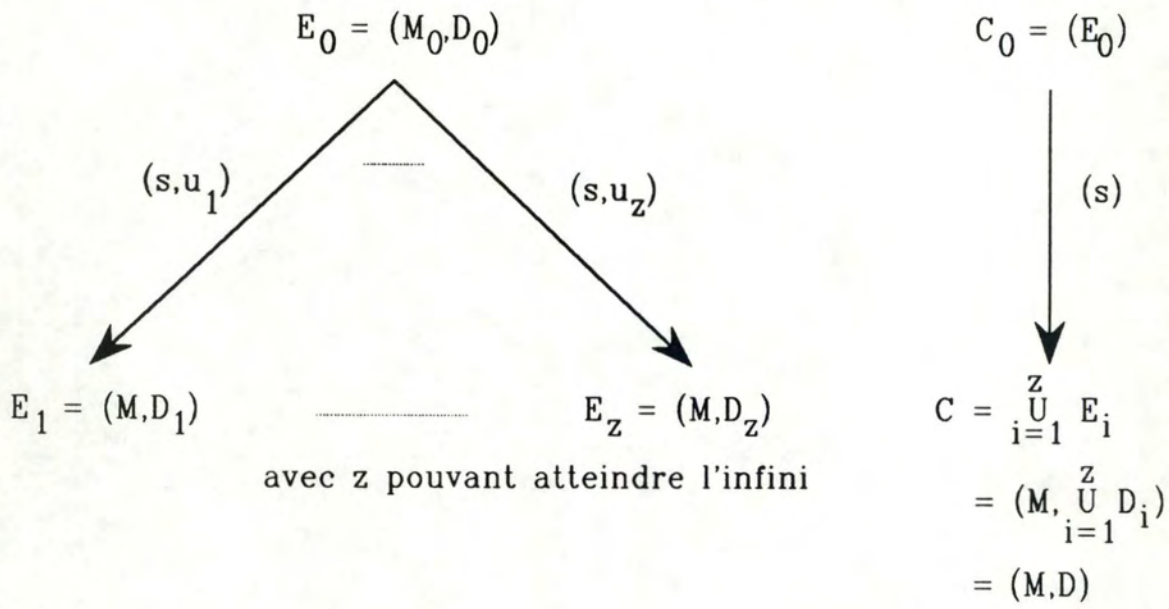


Figure 2.4. Passage de la notion d'état à celle de classe d'états

2.4. Classes d'états et méthode par énumération

[BER 82] [BER 83b] [MEN 82] [ROU 85b]

2.4.1. Notion de domaine de tir

Avant de développer le concept de classes d'états, introduisons un mécanisme plus commode à manipuler que les états, celui de domaine de tir, que l'on note D.

D représente un ensemble de vecteurs possédant une composante pour chaque transition sensibilisée par M et peut être vu comme l'ensemble des solutions du système d'inéquations linéaires comportant une variable pour chaque transition sensibilisée.

Pour l'exemple de la figure 2.1., on aura :

$E_0 = (M_0, D_0)$:

M_0 : $p_1(1)$

I_0 : ensemble des solutions de :

$2 \leq t_1 \leq 3$

$E_1 = (M_1, D_1)$:

M_1 : $p_2(1)$, $p_3(1)$

I_1 : ensemble des solutions de :

$2 \leq t_2 \leq 3$

$3 \leq t_3 \leq 4$

Dorénavant, un état d'un réseau de Petri temporel sera vu comme un couple E = (M,D) où :

. M représente le marquage de l'état

. D représente le domaine de tir

2.4.2. Définition des classes d'états

Dans un premier temps, nous ne considérons que les réseaux de Petri temporels *t-saufs*, c'est-à-dire tels qu'*aucune transition ne peut être plusieurs fois simultanément sensibilisée*.

Une classe d'états associée à une séquence de tir (s) tirable depuis l'état initial est définie (Cf. figure 2.4.) par un couple C = (M,D) où :

. M est le marquage atteint depuis le marquage initial en tirant la séquence s; M est commun à tous les états de la classe.

. D est la réunion des domaines de tir de tous les états accessibles depuis l'état initial en tirant tous les échéanciers (s,u) réalisables pour une séquence de tir s donnée.

Comme par définition, $D = \cup D_i$ où chaque D_i est caractérisé par un système d'inéquations, D peut encore s'exprimer comme étant l'ensemble des solutions d'un système d'inéquations linéaires, avec une variable associée à chaque transition sensibilisée par le marquage M de la classe :

$$D (\subseteq \Theta(M)) = \{ \underline{t} \in \Theta^k \mid A . \underline{t}^T \geq \underline{b} \}$$

où

. $\underline{t}$ est le vecteur ligne instant de tir

. $\underline{b}$ est un vecteur colonne de coefficients

. A est une matrice de k colonnes dont les $A[i,j] \in \{0,1,-1\}$ avec un ou deux éléments non nuls par ligne car chaque équation linéaire a au plus deux variables. Si une ligne a deux éléments non nuls, ils doivent être forcément différents.

. $\Theta(M)$ représente l'espace de tir de M de dimension k

$$\Theta(M) = \underline{t} \in \{ \Theta^k \mid \forall i \in G(M) : t_i \geq DSmin(t_i) \}$$

où

. G(M) est l'ensemble des indice des k transitions sensibilisées par M.

La composante t_i d'un vecteur $\underline{t} \in \Theta(M)$ est donc associée à la transition d'indice i sensibilisée par M.

Exemple :

$C_1 = (M_1, D_1)$ tel que :

$M_1 : p_2(1), p_3(1)$

$D_1 : 2 \leq t_2 \leq 3$
$\quad\quad 3 \leq t_3 \leq 4$

$\underline{t} = (t_2, t_3)$ où $t_2 \in [2,3]$ et $t_3 \in [3,4]$

$A . \underline{t}^T \geq \underline{b}$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} t_2 \\ t_3 \end{pmatrix} \geq \begin{pmatrix} 2 \\ 3 \\ -3 \\ -4 \end{pmatrix}$$

Considérant [MEN 82], toutes les inéquations définissant le domaine de tir sont de trois types possibles :

- les deux premiers définissent les intervalles de tir des transitions sensibilisées :

$$t_i \geq a_i$$
$$t_i \leq b_i$$

- le troisième traduit l'interdépendance entre les instants de tirs des transitions sensibilisées :

$$\underline{t}_i - \underline{t}_j \leq c_{ij} = f°(a_i,b_i,a_j,b_j)$$

2.4.3. Calcul des classes d'états

Avant d'exposer la procédure d'obtention de la classe suivante, présentons la règle de tir des transitions qui permet le calcul des classes d'états de manière récursive de telle sorte que toute classe associée à la séquence s.t soit calculée depuis la classe associée à la séquence s.

A) Règle de tir d'une transition

Une transition t_i est tirable depuis la classe C = (M,D) si, et seulement si, les deux conditions suivantes sont satisfaites :

a) t_i est sensibilisée par le marquage M :

$$\forall p \in P : M(p) \geq Pré(p,t_i)$$

b) Dans le domaine D, il existe un vecteur tel que la composante instant de tir relative à la transition t_i a une valeur inférieure ou égale à celles des autres composantes :

$$\exists \underline{t} \text{ tel que } (A \,.\, \underline{t}^T \geq \underline{b}) \text{ et } (\underline{t}_i \leq \underline{t}_j \; \forall j \neq i)$$

où le système $(A \,.\, \underline{t}^T \geq \underline{b})$ a pour ensemble de solutions le domaine D.

La condition b) complète la condition a) en interdisant le tir de toute transition sensibilisée dont l'intervalle de tir est strictement supérieur à celui d'une autre transition sensibilisée.

L'expression de la deuxième condition est ici plus complexe que dans le cas des états, en ce sens qu'il y a des relations possibles entre les instants de tirs des différentes transitions.

Notons enfin que le tir d'une transition est instantané.

B) Calcul de la classe suivante

Soit la transition t_i tirable à partir de la classe C = (M,D) dans laquelle $D = (A \,.\, \underline{t}^T \geq \underline{b})$.

Le tir de la transition t_i engendre la classe suivante C' = (M',D') qui est caractérisée par :

1. Le marquage M' obtenu comme pour les réseaux de Petri classiques :

$$\forall p \in P : M'(p) = M(p) - Pré(p,t_i) + Post(p,t_{i)}$$

2. Le nouveau domaine D' obtenu en quatre étapes :

2.1. Augmenter le système $A \cdot \underline{t}^T \geq \underline{b}$ avec les conditions de tir de t_i

$$A \cdot \underline{t}^T \geq \underline{b}$$

$$\underline{t}_i \leq \underline{t}_j, \ \forall j \neq i$$

ce qui exprime que t_i est tirée la première parmi les transitions sensibilisées.

2.2. Dans le système précédent, effectuer le changement de variables suivant :

$$\underline{t}_j = \underline{t}_i + \underline{t}_j' \quad \forall j \neq i$$

Le système est transformé comme suit[1] :

$$. \ a_j' \leftarrow \max(0, a_j - b_i, -c_{ij})$$
$$. \ b_j' \leftarrow \min(b_j - a_i, c_{ji})$$
$$. \ c_{kl}' \leftarrow \min(b_k - a_l, c_{kl})$$

Rem. : $-c_{ij}$ résulte de $\underline{t}_i - \underline{t}_j \leq c_{ij}$ et du changement de variable $\underline{t}_j = \underline{t}_i + \underline{t}_j'$; ce qui entraîne : $-\underline{t}_j' = \underline{t}_i - \underline{t}_j \leq c_{ij}$ et $\underline{t}_j' \geq -c_{ij}$

Ensuite, éliminer du système toutes les anciennes variables, $\underline{t}_i$ comprise, pour ne garder que les nouvelles, $\underline{t}_j'$.

On définit ainsi le nouveau domaine de tir pour les transitions restées sensibilisées pendant le tir de t_i; ce domaine a pour nouvelle origine de temps la date à laquelle t_i a été tirée.

2.3. Supprimer par substitution dans le système obtenu les variables associées aux transitions en conflit avec t_i pour le marquage M; ces transitions ont été désensibilisées par le tir de t_i.

Chaque élimination, par exemple de la variable $\underline{t}_e$, correspond à la transformation suivante du système d'inéquations :

$$. \ a_j' \leftarrow \max(a_j, a_e - c_{ej})$$
$$. \ b_j' \leftarrow \min(b_j, b_e + c_{je})$$
$$. \ c_{kl}' \leftarrow \min(c_{kl}, c_{ke} + c_{el})$$
. toute inéquation contenant la variable $\underline{t}_e$ disparaît du système

2.4. Dans ce dernier système, pour chaque transition nouvellement sensibilisée, introduire une variable supplémentaire, contrainte à appartenir à l'intervalle statique de tir de la transition.

---

1 Les formules présentées ici ont été dérivées de la forme canonique en y appliquant successivement les étapes à suivre pour l'obtention du nouveau domaine.

Chaque introduction d'une variable, soit par exemple la variable t_n, conduit à la transformation suivante du système :

. pour tout j, k, l distincts de n, a_j, b_j, c_{kl} ne changent pas

. introduction de la nouvelle variable t_n telle que : $DSmin(t_n) \leq t_n \leq DSMax(t_n)$

. exprimer les relations d'interdépendance entre la variable t_n et les autres variables :

$$t_n - t_l \leq c_{nl} \qquad \forall l \neq n$$
$$t_k - t_n \leq c_{kn} \qquad \forall k \neq n$$

où les valeurs par défaut de c_{nl}, c_{kn} peuvent être choisies telles que :

$$c_{nl} = DSMax(t_n) - a_l$$
$$c_{kn} = b_k - DSmin(t_n)$$

Le nouveau système peut s'écrire $A' \cdot \underline{t}'^T \geq \underline{b}'$

Illustration

Si nous appliquons la méthode d'analyse présentée ci-dessus au réseau de Petri temporel de la figure 2.1. nous obtenons 11 classes d'états :

$C_0 = (M_0, D_0)$:
$M_0 : p_1(1)$
$D_0 : 2 \leq t_1 \leq 3$

$C_1 = (M_1, D_1)$:
$M_1 : p_2(1), p_3(1)$
$D_1 : 2 \leq t_2 \leq 3$
$3 \leq t_3 \leq 4$

$C_2 = (M_2, D_2)$:
$M_2 : p_3(1), p_4(1)$
$D_2 : 0 \leq t_3 \leq 2$
$0 \leq t_4 \leq 0$

$C_3 = (M_3, D_3)$:
$M_3 : p_4(1), p_5(1)$
$D_3 : 0 \leq t_4 \leq 0$
$0 \leq t_5 \leq 0$

$C_4 = (M_4, D_4)$:
$M_4 : p_5(1), p_6(1)$
$D_4 : 0 \leq t_5 \leq 0$

$C_5 = (M_5, D_5)$:
$M_5 : p_6(1), p_7(1)$
$D_5 : 3 \leq t_6 \leq 3$

$C_6 = (M_6, D_6)$:
$M_6 : p_8(1)$
$D_6 : 0 \leq t_7 \leq 0$

$C_7 = (M_7, D_7)$:
$M_7 : p_4(1), p_7(1)$
$D_7 : 0 \leq t_4 \leq 0$

$C_8 = (M_8, D_8)$:
$M_8 : p_3(1), p_6(1)$
$D_8 : 0 \leq t_3 \leq 2$

$C_9 = (M_9, D_9)$:
$M_9 : p_2(1), p_5(1)$
$D_9 : 0 \leq t_2 \leq 0$
$0 \leq t_5 \leq 0$

$C_{10} = (M_{10}, D_{10})$:
$M_{10} : p_2(1), p_7(1)$
$D_{10} : 0 \leq t_2 \leq 0$

C) Egalité de deux classes d'états

Deux classes d'états $C_1 = (M_1,D_1)$ et $C_2 = (M_2,D_2)$ sont égales par définition si, et seulement si, leurs marquages et leurs domaines respectifs sont égaux.

Comparer les deux domaines D_1 et D_2 revient à comparer les deux systèmes d'inéquations linéaires définissant D_1 et D_2. Mais cette manière de faire peut être astreignante !

Aussi, le domaine de tir dans toute classe d'états pouvant être exprimé comme l'ensemble des solutions d'un système d'inéquations linéaires avec au plus deux variables par inéquation,

$$a_i \leq \underline{t}_i \leq b_i \qquad \forall\, i$$
$$\underline{t}_i - \underline{t}_j \leq c_{ij} \qquad \forall\, j \neq i$$

la procédure de comparaison peut être simplifiée en réécrivant les inéquations sous leur forme canonique :

$$a_i^* \leq \underline{t}_i \leq b_i^* \qquad \forall\, i$$
$$\underline{t}_i - \underline{t}_j \leq c_{ij}^* \qquad \forall\, j \neq i$$

avec $a_i^* = \min(\underline{t}_i, \forall\, \underline{t} \in D)$
$b_i^* = \max(\underline{t}_i, \forall\, \underline{t} \in D)$
$c_{ij}^* = \max(\underline{t}_i - t_j\ , \forall\, \underline{t} \in D)$

où $\underline{t}$ est le vecteur courant du domaine de tir D et $\underline{t}_i$ la composante du vecteur $\underline{t}$ associée à la transition t_i.

Pour démontrer l'égalité de deux domaines, il suffira de vérifier l'identité de leur forme canonique.

D) Construction du graphe des classes d'états

La relation d'accessibilité définie par la règle de tir permet de générer un *arbre de classes d'états* : sa racine est la classe initiale, ses sommets sont les classes d'états. Un arc étiqueté avec t_i relie la classe C à la classe C' si, et seulement si, la transition t_i est tirable depuis la classe C et son tir depuis C génère C'. Chaque nouvelle classe est comparée avec celles existantes et, s' il y a égalité, l'exploration de la branche concernée est abandonnée, ce qui évite une explosion combinatoire.

Le *graphe des classes d'états* est obtenu à partir de cet arbre en réunissant les sommets portant des classes égales.

L'existence d'un chemin entre la classe initiale et la classe C implique qu'un échéancier de tir depuis l'état initial jusqu'à un état appartenant à la classe C est réalisable.

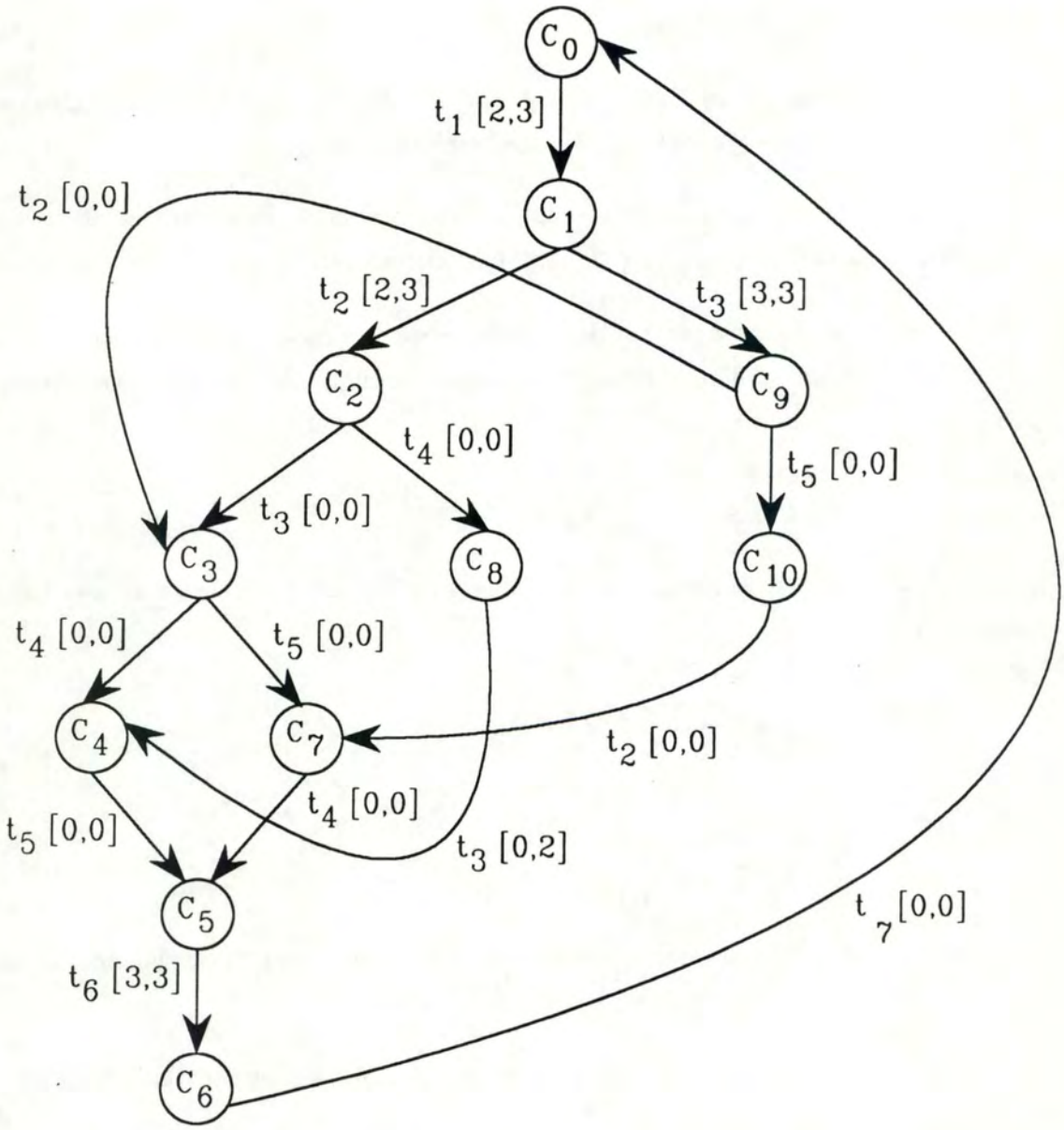


Figure 2.5. Graphe des classes d'états

Illustration

L'ensemble complet des scénarios d'évolution du réseau décrit par la figure 2.1. peut être exprimé convenablement par le graphe des classes d'états représenté par la figure 2.5.

Chaque arc du graphe peut également être décrit textuellement de la manière suivante :

$C_0 \rightarrow t_1 \in [2,3] / C_1$
$C_1 \rightarrow t_2 \in [2,3] / C_2, t_3 \in [3,3] / C_9$
$C_2 \rightarrow t_3 \in [0,0] / C_3, t_4 \in [0,0] / C_8$
$C_3 \rightarrow t_4 \in [0,0] / C_4, t_5 \in [0,0] / C_7$
$C_4 \rightarrow t_5 \in [0,0] / C_5$
$C_5 \rightarrow t_6 \in [3,3] / C_6$
$C_6 \rightarrow t_7 \in [0,0] / C_0$
$C_7 \rightarrow t_4 \in [0,0] / C_5$
$C_8 \rightarrow t_3 \in [0,2] / C_4$
$C_9 \rightarrow t_2 \in [0,0] / C_3, t_5 \in [0,0] / C_{10}$
$C_{10} \rightarrow t_2 \in [0,0] / C_7$

L'interprétation de la première ligne, par exemple, se fait comme suit : le tir de t_1 depuis la classe C_0, à un instant compris dans l'intervalle [2,3], conduit à la classe C_1.

Notons que l'examen du contenu des classes d'états et des échéanciers de tir sur le graphe, montre que le réseau est borné, vivant et réinitialisable.

E) Remarques

L'utilité de la méthode d'énumération des classes d'états n'est réelle que si le réseau est borné. Or, les propriétés d'accessibilité et de borné ne sont pas décidables pour les réseaux de Petri temporels [MEN 82]. Cela complique la méthode d'analyse du modèle par énumération des classes d'états.

Au chapitre suivant, nous verrons que la seule solution envisageable consiste à établir des conditions suffisantes mais non nécessaires permettant un contrôle efficace de l'énumération.

2.4.4. Cas des transitions multi-sensibilisées

Jusqu'ici, dans un but de simplification, nous n'avons considéré que les réseaux dans lesquels toute transition est simplement sensibilisée, c'est-à-dire les réseaux tels que pour tout marquage M et transition t_i, on a, pour une place p au moins, $M(p) < 2$. Pré(p,t_i).

Mais, qu'en est-il dans le cas de transitions multi-sensibilisées ?

Considérons la transition t_i, d'intervalle de tir initial $[a_i,b_i]$ et sensibilisée par M depuis un temps θ (son intervalle de tir actuel est alors $[\max(0,a_i-\theta),b_i-\theta]$) et supposons que le tir d'une autre transition, soit t_j avec $j \neq i$, conduise à un nouveau marquage M' tel que t_i soit sensibilisée une deuxième fois.

On a alors :

- $\forall p \in P : M'(p) \geq 2 . Pré(p,t_i)$ et $\exists p \in P : M'(p) < 3 . Pré(p,t_i)$
- t_i possède deux intervalles de tir :
  - $[\max(0,a_i-\theta),b_i-\theta]$ relatif à la première sensibilisation de t_i
  - $[a_i,b_i]$ relatif à la deuxième sensibilisation de t_i

Le problème se résume à savoir lequel de ces intervalles sera considéré lorsque t_i sera tirée ou lorsque t_i sera en conflit avec une autre transition.

Nous pouvons envisager plusieurs choix : l'un quelconque de ces intervalles, le plus ancien ou le plus récent.

La solution qui a été retenue [ROU 85b] est de tirer en priorité, parmi les instances d'une transition multi-sensibilisée, la plus ancienne. Ce choix a été guidé par la sémantique même d'une transition, à savoir que lorsque plusieurs événements identiques surviennent, le traitement se fait dans l'ordre d'apparition.

Il est alors possible d'étendre l'algorithme d'énumération des classes d'états aux réseaux temporels avec transitions multi-sensibilisées de la manière suivante : aux étapes 2.1., 2.2., 2.3. de la transformation du domaine de tir, on ne fait plus référence à la transition t_i, mais à la plus ancienne occurrence de la transition t_i, les diverses occurrences de t_i étant indiquées par les variables indicées avec le numéro j de l'occurrence, soit $t_{i,j}$.

2.5. Conclusion

Nous venons de voir une méthode d'analyse pour les réseaux de Petri temporels.

L'infinité possible des états accessibles nous a amené à considérer la notion de classes d'états.

Nous avons montré que les domaines de tir associés à chaque classe d'états pouvaient être exprimés comme un système d'inéquations contenant une variable pour chaque transition sensibilisée par le marquage de la classe.

De plus, une règle de tir a été présentée pour construire de manière récursive l'ensemble des classes d'états d'un réseau de Petri temporel donné.

Après avoir montré comment déterminer l'égalité entre deux classes d'états, nous avons donné un moyen de représenter l'ensemble des classes et leurs relations d'accessibilité; il s'agit du graphe des classes des états accessibles qui permet de représenter le comportement des systèmes modélisés.

Si le graphe est fini, le comportement du système ainsi représenté peut être vérifié en étudiant ses propriétés : analyse de connectivité, recherche de chemins, informations contenues dans les sommets, ... sinon, l'analyse se réduit à une simulation du fonctionnement partiel du réseau.

Pour compléter cette analyse, nous donnerons au chapitre suivant une condition suffisante pour déterminer si le graphe des états accessibles d'un réseau de Petri temporel est fini.

CHAPITRE 3

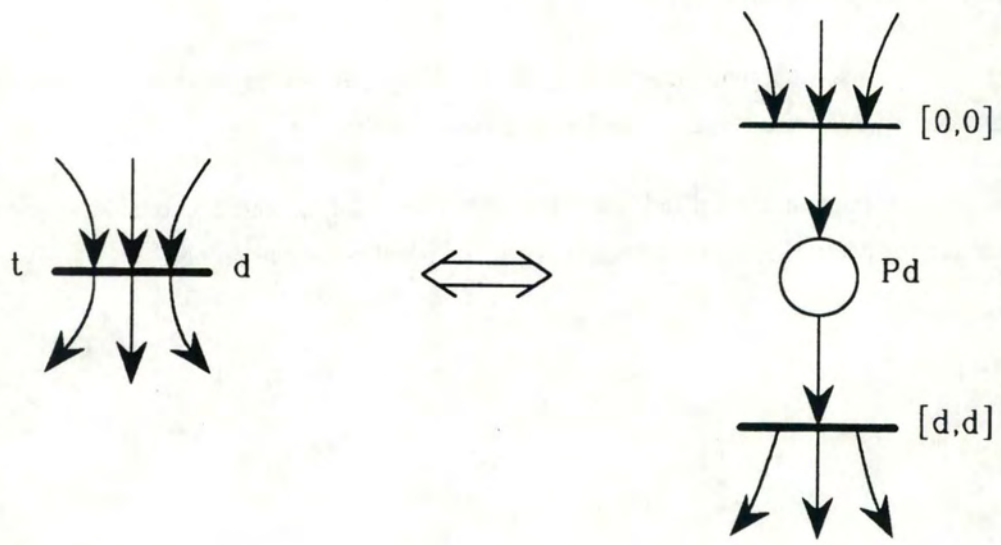
PUISSANCE ET PROPRIETES DU MODELE DES RESEAUX DE PETRI TEMPORELS

3.1. Introduction

Dans la première partie de ce chapitre, nous donnons les relations existantes entre les diverses classes des réseaux de Petri et les réseaux de Petri temporels, de même que des méthodes de transformation qui permettent de simuler la plupart des extensions des réseaux de Petri par des réseaux temporels équivalents.

Dans une deuxième partie, nous présentons les propriétés spécifiques du modèle et mettons en évidence certains problèmes d'indécidabilité.

Ces problèmes d'indécidabilité nous amènent finalement à considérer une condition suffisante pour déterminer si un réseau de Petri temporel est borné ou non.



Modèle de Ramchandani

Modèle de Merlin

Figure 3.1.

3.2. Puissance d'expression du modèle des réseaux de Petri temporels

3.2.1. Simulation du comportement des réseaux de Petri classiques

Les réseaux de Petri classiques sont des cas particuliers des réseaux de Petri temporels.

Ce qui différencie les réseaux de Petri classiques des réseaux de Petri temporels, c'est que lorsque plusieurs transitions sont sensibilisées par un marquage, il n'existe pas, dans les réseaux classiques, de priorité pour le tir des transitions; tandis que pour les réseaux temporels, le fait d'ajouter un intervalle de tir à chaque transition peut conduire à des priorités.

Ainsi, s'il est possible d'associer à chaque transition un intervalle de temps tel qu'il n'introduise aucune contrainte sur les instants de tir, nous pourrons représenter tout réseau classique par un réseau temporel.

Pour que t_i soit prioritaire à t_j il suffit que $b_i < a_j$; car si t_i doit être tirée avant b_i et t_j doit être tirée après a_j, t_j ne peut pas être tirée avant t_i.

Dès lors, une solution triviale pour ne pas introduire de priorité est celle qui associe à chaque transition l'intervalle $[0, \infty]$.

Le même effet est également possible, si à la place de l'intervalle $[0, \infty]$, nous associons aux transitions des intervalles différents mais qui toutefois respectent la contrainte suivante :

. soit $DSmin(t_i)=0 \qquad \forall t_i : 1 \leq i \leq n$

. soit $DSMax(t_i)=\infty \qquad \forall t_i : 1 \leq i \leq n$

Ces assignations d'intervalles n'introduisent aucune priorité entre les transitions.

3.2.2. Simulation des réseaux de Petri temporisés

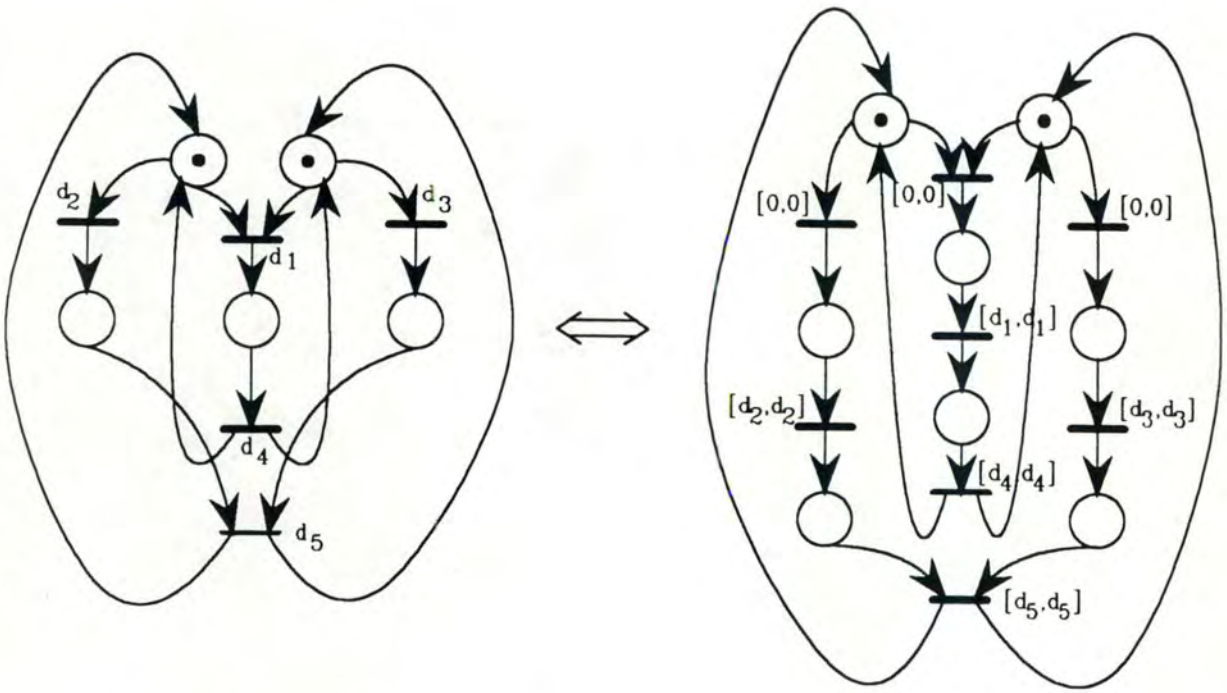
Tout réseau de Petri temporisé peut être converti en un réseau de Petri temporel qui engendre le même fonctionnement [MEN 82] [ROU 85b].

A) Le modèle temporisé de Ramchandani

Le modèle de Ramchandani [RAM 74] associe à chaque transition une durée représentant le temps que dure le franchissement de la transition. Pendant ce temps, les jetons en entrée de la transition sont indisponibles.

Considérons une transition quelconque de durée d et effectuons les transformations pour arriver à une représentation sous la forme du modèle temporel de Merlin (Cf. figure 3.1.).

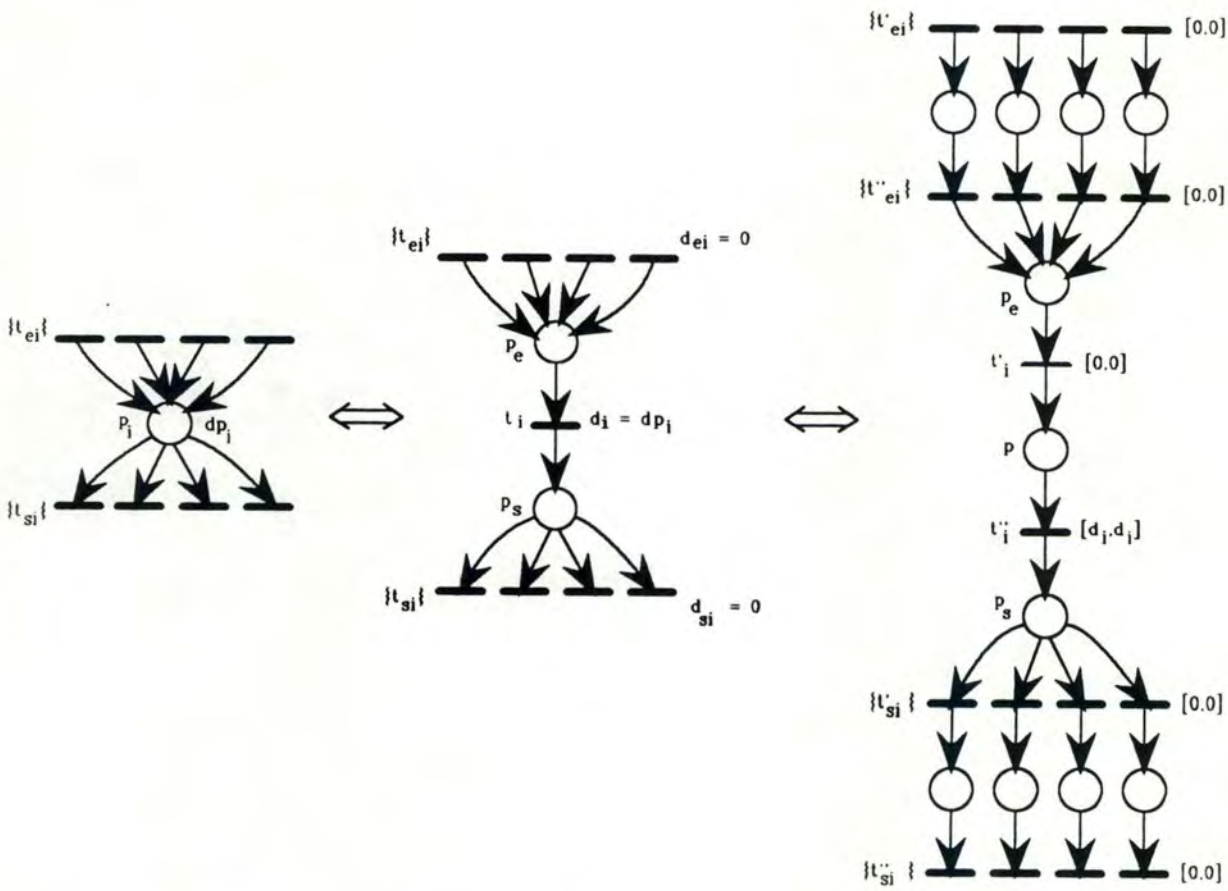
Dans le modèle de Merlin, la place p_d simule la réservation des jetons opérée par le tir de t pendant le temps d.



Modèle de Ramchandani

Modèle de Merlin

Figure 3.2.

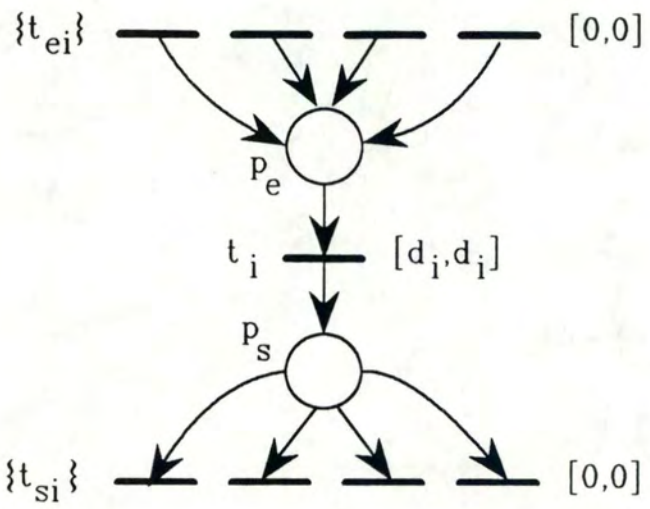


Modèle de Sifakis

Modèle de Ramchandani

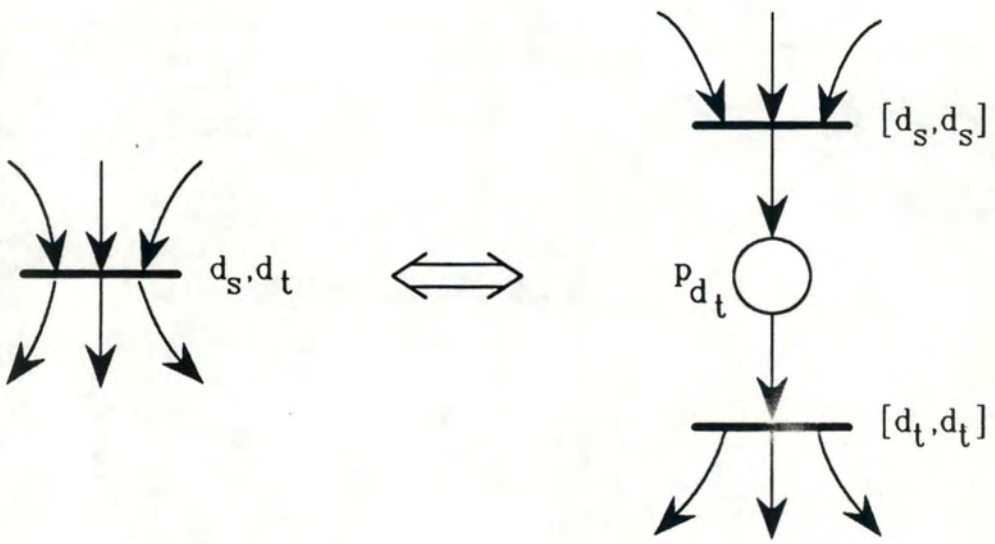
Modèle de Merlin

Figure 3.3.



Modèle de Merlin réduit

Figure 3.4.



Modèle de Razouk

Modèle de Merlin

Figure 3.5.

Dans le cas où toute place d'entrée de la transition n'a qu'un seul arc de sortie, la substitution du paramètre temporel d par l'intervalle de temps [d,d], sans modifier le graphe, suffit pour passer du modèle temporisé de Ramchandani au modèle temporel de Merlin (Cf. figure 3.2.).

Pour les autres transitions, il est impératif de faire la substitution présentée à la figure 3.1.

C) Le modèle temporisé de Sifakis

Le modèle temporisé de Sifakis [SIF 77] associe à chaque place du réseau une durée de séjour. Cette durée correspond au temps pendant lequel les jetons arrivant dans la place sont indisponibles avant de pouvoir être utilisés pour tirer les transitions.

Sifakis [SIF 79] a montré l'équivalence de son modèle avec celui de Ramchandani, de sorte qu'en passant par le modèle de Ramchandani, nous pouvons transformer le modèle de Sifakis en un modèle temporel de Merlin (Cf. figure 3.3.).

Notons, que dans le modèle de Ramchandani, t_i a une durée de franchissement égale, dans le modèle de Sifakis, à la durée de séjour associée à la place p_i, soit dp_i.

Etant donné que pour toute place d'entrée d'une transition qui n'a qu'un seul arc de sortie il suffit, pour passer du modèle de Ramchandani au modèle de Merlin, de substituer le paramètre temporel d par l'intervalle [d,d], le réseau de la figure 3.3. peut être réduit de la manière présentée à la figure 3.4.

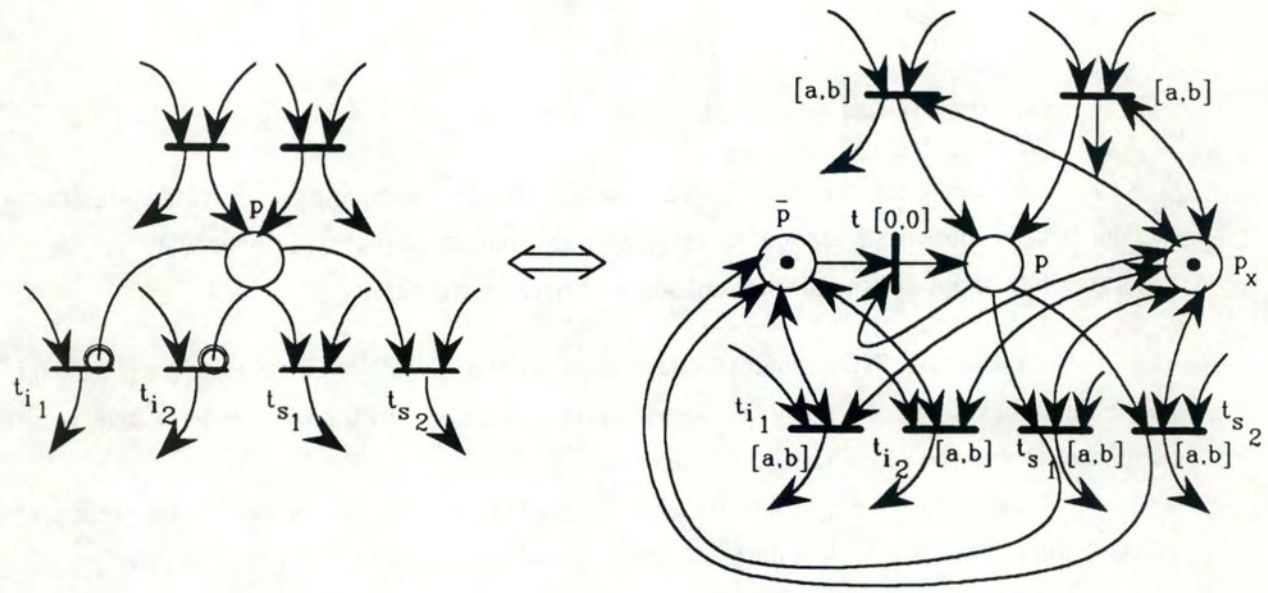
C) Le modèle temporisé de Razouk

Le modèle proposé par Razouk [RAZ 83] associe deux paramètres temporels à chaque transition :

- une durée de sensibilisation, d_s, traduisant le fait que la transition doit rester sensibilisée un temps d_s avant de pouvoir être tirée,
- une durée de tir, d_t, indiquant le temps de franchissement de la transition.

Précisons encore qu'une transition ayant démarré son tir ne peut plus être désensibilisée.

La conversion du modèle de Razouk en un réseau de Petri temporel est obtenue grâce aux transformations exprimées par la figure 3.5.



Réseau de Petri avec arcs inhibiteurs

Modèle de Merlin

Figure 3.6.

3.2.3. Simulation des réseaux avec arcs inhibiteurs

Rappelons qu'un arc inhibiteur est un arc qui relie une place à une transition et qui empêche le tir de cette transition si le marquage de la place est non nul [BRA 83b].

Les arcs inhibiteurs se distinguent des autres arcs par le fait qu'ils sont terminés par un rond au lieu d'une flèche.

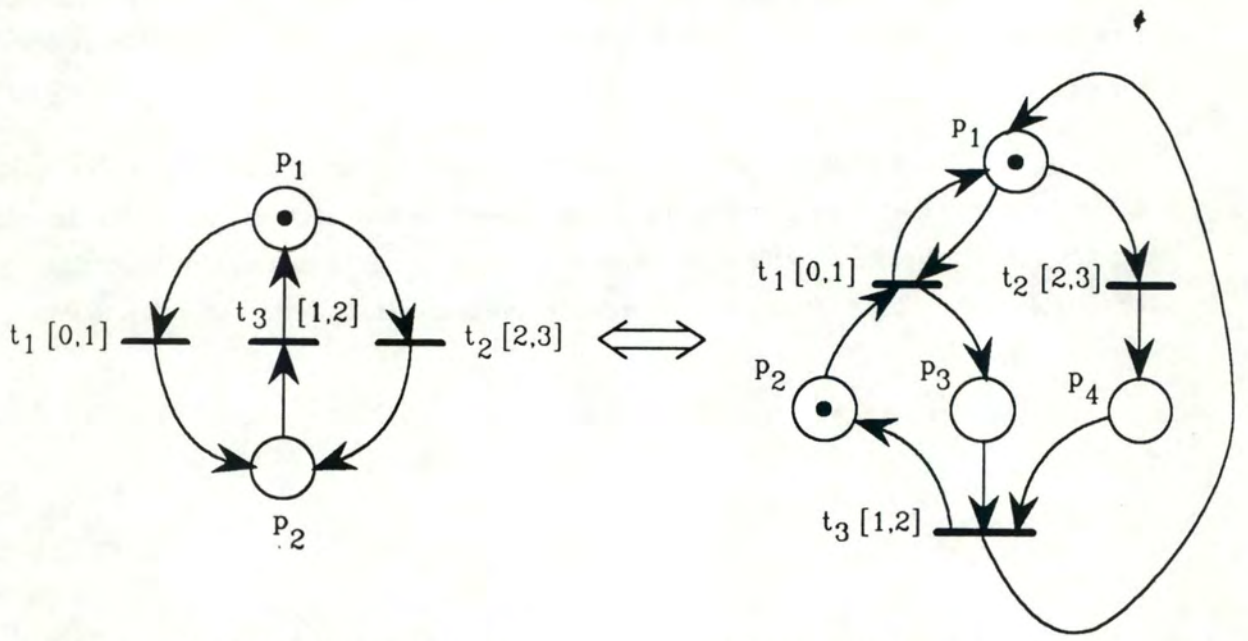
Tout réseau de Petri avec arcs inhibiteurs peut être décrit par un réseau de Petri temporel qui engendre le même fonctionnement.

Il suffit d'appliquer à toute transition destination d'un arc inhibiteur la transformation représentée par la figure 3.6.

Tant que $M(p) = 0$, $t_{i.j}$ est sensibilisée. Lorsque $M(p) \geq 1$, la transition t devient prioritaire par rapport aux autres transitions et le tir de t désensibilise les transitions $t_{i.j}$ tant que $M(p) \geq 1$.

Le fait de rajouter la place p_x telle que $M(p_x) = 1$ avec des arcs élémentaires pour toute transition différente de la transition t ne change en rien le comportement du réseau initial mais permet que toute transition sensibilisée soit réarmée après le tir de n'importe quelle autre transition.

Si maintenant le même intervalle [a,b] est associé à chaque transition, à l'exception toutefois de la transition t, aucune nouvelle priorité n'est introduite entre ces transitions car elles ont toutes une même place d'entrée et de sortie en commun, p_x, et toute transition sensibilisée est donc réarmée après chaque tir; ce qui signifie que les intervalles de tir restent tous identiques.



Réseau de Petri classique vivant

Réseau de Petri temporel non vivant

Réseau de Petri classique non vivant

Réseau de Petri temporel vivant

Figure 3.7.

3.3. Propriétés spécifiques

Avant toute chose, redéfinissons les propriétés de réseau bloquant, vivant et réinitialisable des réseaux de Petri temporels, propriétés qui sont d'ailleurs similaires à celles des réseaux de Petri classiques.

3.3.1. Définition 1

Un *réseau de Petri temporel* R_Θ est dit *bloquant* dans un état $E = (M,D)$ accessible dans R_Θ à partir de l'état initial E_0 si, et seulement si, aucune transition n'est sensibilisée par M.

3.3.2. Définition 2

Un *réseau de Petri temporel* R_Θ est dit *vivant* si, et seulement si, quel que soit l'état accessible $E_1 = (M_1,D_1)$ à partir de l'état initial $E_0 = (M_0,D_0)$ et quel que soit $t \in T$, il existe une séquence tirable de transitions, s, à partir de E_1 telle que :

$$E_1 = (M_1,D_1) \xrightarrow{s} E_2 = (M_2,D_2)$$

et

t est tirable en E_2

3.3.3. Définition 3

Un *réseau de Petri temporel* R_Θ est dit *cyclique ou réinitialisable* à partir d'un état initial E_0 si, et seulement si, quel que soit l'état E_1 accessible à partir de E_0, il existe une séquence tirable de transitions, s, à partir de E_1 telle que :

$$E_1 = (M_1,D_1) \xrightarrow{s} E_0 = (M_0,D_0)$$

Comme nous pouvons le constater, ces propriétés se définissent de la même manière que pour les réseaux de Petri classiques.

Notons toutefois qu'il n'existe aucune relation entre la vivacité d'un réseau de Petri temporel et celle du réseau de Petri classique sous-jacent, ainsi que l'illustre la figure 3.7.

3.3.4. Indécidabilité de certaines propriétés

A) Théorème 1

Les propriétés d'accessibilité et de borné ne sont pas décidables pour les réseaux de Petri temporels.

Dans [BRA 83b], il a été démontré pour les réseaux de Petri avec arcs inhibiteurs que les propriétés d'accessibilité et de borné sont indécidables.

Comme tout réseau de Petri avec arcs inhibiteurs peut être décrit par l'intermédiaire des réseaux de Petri temporels, il s'ensuit que ces deux propriétés sont également non décidables pour ceux-ci.

La preuve directe de l'indécidabilité de ces deux propriétés a été produite par Jones et al. [JON 77] qui a montré directement la correspondance entre les réseaux de Petri temporels et les automates à deux compteurs.

En conséquence, puisque les classes d'états sont des couples (marquage, domaine de tir), nous pouvons dire que la finitude des classes d'états d'un réseau de Petri temporel est indécidable.

Il va de soi que ce qui vient d'être dit complique la méthode d'analyse par énumération des classes d'états, d'où l'importance des deux théorèmes qui suivent.

B) Théorème 2

Si les dates statiques de tir au plus tôt et les dates statiques de tir au plus tard affectées aux transitions sont des nombres rationnels, alors le nombre de classes d'états d'un réseau temporel t-sauf est fini ssi le réseau est borné [BER 82] [MEN 82].

Rem. : la restriction des bornes statiques de tir à des nombres rationnels, essentielle pour démontrer le théorème, n'admet aucune limite pratique à l'utilisation des réseaux de Petri temporels car tout nombre réel peut être approximé par un nombre rationnel.

■ *Lemme 1*

Les domaines de tir des classes d'états pour tout réseau t-sauf peuvent être exprimés comme l'ensemble de solutions des systèmes d'inéquations

$$a_i \leq \underline{t}_i \leq b_i \quad \forall i$$
$$\underline{t}_k - \underline{t}_l \leq c_{kl} \quad \forall k,l \quad avec\ k \neq l$$

où $\underline{t}_i$ est la variable associée à la transition t_i sensibilisée par le marquage de la classe.

Démonstration

Le domaine initial peut être décrit comme un ensemble d'inéquations de la forme

$$DSmin(t_i) \leq \underline{t}_i \leq DSMax(t_i)$$
$$\underline{t}_k - \underline{t}_l \leq c_{kl} \quad \text{avec } c_{kl} = DSMax(t_k) - DSmin(t_l)$$

où la variable $\underline{t}_i$ est associée à la transition t_i sensibilisée par le marquage.

Or, chacune des transformations du système, induite par la règle de tir définie au chapitre précédent, conserve la forme générale du système d'inéquations définissant le domaine de tir.

Dès lors, tous les systèmes d'inéquations représentant les différents domaines de tir ont la même forme générale et le lemme est ainsi vérifié.

■ *Lemme 2*

Les constantes a_i, b_i, c_{kl} de tout système calculé depuis l'état initial par la règle de tir sont des combinaisons linéaires à coefficients entiers des bornes statiques inférieures et supérieures associées aux transitions du réseau temporel.

On peut encore écrire :

$\forall i \ \exists \lambda_1, \ldots, \lambda_{2n} \in Z$

$a_i = \lambda_1 DSmin(t_1) + .. + \lambda_n DSmin(t_n) + \lambda_{n+1} DSMax(t_1) + \ldots + \lambda_{2n} DSMax(t_n)$

(de même avec b_i et c_{kl})

Démonstration

La preuve est directe : le lemme 2 est vrai pour le système initial et cette propriété est conservée par les différentes étapes de la règle de tir.

■ *Lemme 3*

Les constantes a_i, b_i, c_{kl} ($\forall$ i, k, l) de tout système calculé à partir du système initial par la règle de tir ont les bornes suivantes :

a) $0 \leq a_i \leq DSmin(t_i)$

b) $0 \leq b_i \leq DSMax(t_i)$

c) $-DSmin(t_l) \leq c_{kl} \leq DSMax(t_k)$

Démonstration

1. Démontrons a) et b)

Pour passer d'un état E à son suivant E',

. si t_i n'était pas sensibilisée dans l'état E mais le devient dans l'état E', alors selon la règle de tir :

$$a_i(E') = DSmin(t_i)$$
$$b_i(E') = DSMax(t_i) \quad (1)$$

. si t_i était sensibilisée dans l'état E et le reste en E' alors, a_i et b_i ne peuvent que décroître. En effet, puisque d'une part, un certain temps (éventuellement nul) s'est écoulé avant de passer de l'état E à l'état E', soit θ ce temps, et que d'autre part, la nouvelle origine des temps correspond à l'instant où le nouvel état a été atteint, il faut retirer un temps θ à a_i et b_i par rapport à l'état E. (2)

Par **(1)** et **(2)**, on déduit que :

$$0 \leq a_i \leq DSmin(t_i)$$
$$0 \leq b_i \leq DSMax(t_i)$$

2. Démontrons c)

. Supposons que dans un état E = (M,D), $c_{kl} > DSMax(t_k)$. Par définition, $c_{kl} = \min \{ \tau_k - \tau_l \mid \forall \tau \in D \}$.

Ceci implique que dans l'état E, il existe $\tau \in D$ tel que :

$$\tau_k - \tau_l > DSMax(t_k) \text{ et donc que } \tau_k > DSMax(t_k) + \tau_l$$

ce qui est contradictoire avec la relation $0 \leq b_k \leq DSMax(t_k)$ donnée par le point b).

On a donc bien $c_{kl} \leq DSMax(t_k)$ (3)

. Comme $t_k - t_l \leq c_{kl}$ (propriété de la forme canonique), à partir du point a) nous avons :

$$0 - DSmin(t_l) \leq t_k - t_l$$
$$\text{ou encore } -DSmin(t_l) \leq c_{kl} \quad (4)$$

Par **(3)** et **(4)**, on déduit aisément que :

$$-DSmin(t_l) \leq c_{kl} \leq DSMax(t_k)$$

Notons que si $DSMax(t_k)$ est non bornée pour t_k, alors b_k et c_{kl} ($\forall l : k \neq l$) sont initialement et restent non bornés.

■ *Lemme 4*

Soient A et B des constantes réelles bornées et $q_1, \ldots, q_n$ un ensemble fini de constantes rationnelles.

Le nombre de combinaisons linéaires à coefficients entiers sur les nombres $q_1, \ldots, q_n$ existant entre les nombres A et B (intervalle limité) est fini.

Autrement dit, le nombre de nombres rationnels X tel que

$$X = \lambda_1 q_1 + \ldots + \lambda_n q_n$$
$$et \quad \lambda_1, \ldots, \lambda_n \in Z$$
$$et \quad q_1, \ldots, q_n \in Q$$
$$et \quad A \leq X \leq B$$

est fini.

Démonstration

Notons d le dénominateur commun des nombres rationnels $q_1, \ldots, q_n$ et Q_i le produit $d.q_i$. Le problème ci-dessus revient à montrer que le nombre de combinaisons linéaires à coefficients entiers des nombres $Q_1, \ldots, Q_n$, entre les bornes d.A et d.B est fini, ce qui est évidemment vrai.

Démonstration du théorème 2

Les constantes a_i, b_i, c_{kl} qui apparaissent dans les systèmes d'inéquations qui décrivent les domaines sont des combinaisons linéaires à coefficients entiers des bornes statiques inférieures et supérieures (lemmes 1 et 2). D'autre part, ces constantes sont soit non bornées et restent alors non bornées, soit ont des bornes inférieures et supérieures qui ne dépendent que des bornes statiques de tir (lemme 3).

De plus, par définition d'un réseau de Petri temporel, les dates statiques de tir au plus tôt et au plus tard sont des nombres rationnels. Ainsi, si nous nous référons au lemme 4, le nombre de constantes a_i, b_i, c_{kl} qui peuvent être calculées par application de la règle de tir est fini.

Ensuite, le réseau étant t-sauf, les systèmes calculés ont nécessairement un nombre fini de variables; il s'ensuit que le nombre de domaines de tir distincts que l'on peut calculer en utilisant la règle de tir est fini.

Enfin, le réseau étant t-sauf et borné, il admet à la fois un nombre fini de marquages (conséquence de la propriété borné) et un nombre fini de domaines de tir distincts. Le nombre de classes d'états est dès lors borné.

Condition suffisante :

Suite au théorème précédent, nous pouvons dire que toute condition suffisante pour la propriété borné fournira également une condition suffisante pour prouver la finitude de l'ensemble des classes d'états.

Le théorème qui suit donne une condition suffisante qui s'est avérée bien adaptée pour les applications que nous envisageons [ROU 85b].

C) Théorème 3

Un réseau de Petri temporel t-sauf est borné s'il n'existe pas deux classes d'états $C = (M,D)$ *et* $C' = (M',D')$ *accessibles depuis la classe initiale* C_0 *telles que :*

1. *C' est accessible depuis C*
2. $\forall p \in P, M'(p) \geq M(p)$
   et
   $\exists p \in P \mid M'(p) > M(p)$
3. $D' = D$

Démonstration

Supposons que le réseau de Petri temporel soit t-sauf et non borné.

S'il est non borné, le réseau temporel admet une séquence de tir de longueur infinie s' passant par une séquence s de classes d'états distinctes deux à deux.

Le réseau étant t-sauf, il admet seulement un nombre fini de domaines de tir distincts (Cf. démonstration théorème 2).

Puisque le réseau contient un nombre fini de domaines de tir distincts et qu'une classe est formée du couple (marquage,domaine), la séquence s contient nécessairement une séquence

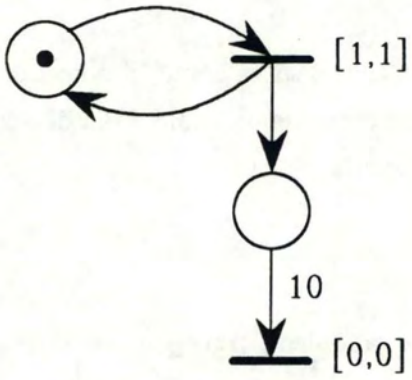


Figure 3.8.

non bornée s' pour laquelle toutes les classes ont le même domaine mais dont les marquages sont différents.

En se référant à [KAR 69], cette séquence s' contient nécessairement deux classes C = (M,D) et C' = (M',D') telles que les trois conditions du théorème soient satisfaites et le théorème est ainsi démontré.

La condition suffisante donnée ici par le théorème 3 complète l'algorithme d'analyse par énumération des classes d'états donné au chapitre précédent.

Soulignons le fait que si la condition est suffisante pour vérifier qu'un réseau est borné, elle n'est pas nécessaire. Pour s'en convaincre, considérons le réseau temporel représenté à la figure 3.8. et pour lequel la condition n'est pas satisfaite et cela bien que le réseau soit borné puisqu'il admet seulement 11 classes d'états.

Finalement, quand le réseau de Petri est borné, il est possible, en utilisant le graphe des classes d'états accessibles, de vérifier les propriétés spécifiques (absence de blocage, terminaison, ...) qui caractérisent le comportement correct du système modélisé, de la même manière que pour les réseaux de Petri classiques avec le graphe des marquages.

3.3. Conclusion

Dans ce chapitre, nous avons montré qu'un réseau de Petri temporel tel que défini par Merlin pouvait simuler tout réseau de Petri classique, aussi bien que les réseaux temporisés de Ramchandani, de Sifakis ou encore de Razouk et les réseaux avec arcs inhibiteurs.

Nous avons également insisté sur le fait que la propriété borné pour les réseaux de Petri temporels est indécidable.

Or, la technique d'analyse par énumération des classes d'états présentées au chapitre précédent, est tributaire de cet aspect, notamment en ce qui concerne la condition d'arrêt de la méthode.

C'est pourquoi, nous avons présenté une condition suffisante permettant de vérifier la propriété borné pour les réseaux de Petri temporels. Celle-ci s'est avérée adéquate pour la plupart des applications pratiques comme par exemple dans le domaine de l'analyse des protocoles de communication.

CHAPITRE 4

LES OUTILS RESEAUX DE PETRI TEMPORELS

4.1. Introduction

A l'heure actuelle, nous pouvons dire que l'utilisation des réseaux de Petri temporels est considérablement freinée par le manque d'outils logiciels qui prennent en compte la variable temps.

L'existence de ces outils est pourtant nécessaire pour assister l'utilisateur lors de la modélisation et l'analyse de systèmes importants.

De tels outils devraient permettre aux utilisateurs de produire rapidement des résultats fiables et de qualité (construction automatique du graphe des classes d'états, étude des propriétés du réseau, ...) et cela, malgré la complexité possible de la description du réseau.

Dans ce chapitre, nous donnons la démarche que nous avons suivie pour l'élaboration d'un outil permettant l'analyse des réseaux de Petri temporels et dont le comportement est basé sur l'approche par énumération des classes d'états. Vu le temps dont nous disposions, il nous était impossible de réaliser l'intégralité de ce travail, c'est pourquoi, parmi les différents éléments que nous présentons dans cette section, nous nous sommes limité, d'un point de vue programmation, à l'aspect édition et simulation.

Ensuite, nous présentons un outil professionnel d'aide à la conception et à l'évaluation de systèmes par réseaux de Petri, développé sous le nom de RdP, par la société Vérilog de Toulouse. Nous avons eu l'occasion de tester cet outil à la fin de notre stage au Laboratoire d'Automatique et d'Analyse des Systèmes de Toulouse. Il nous a permis de réaliser l'analyse du

protocole de scrutation cyclique de la couche liaison de données du réseau FIP, qui sera présentée au chapitre 6.

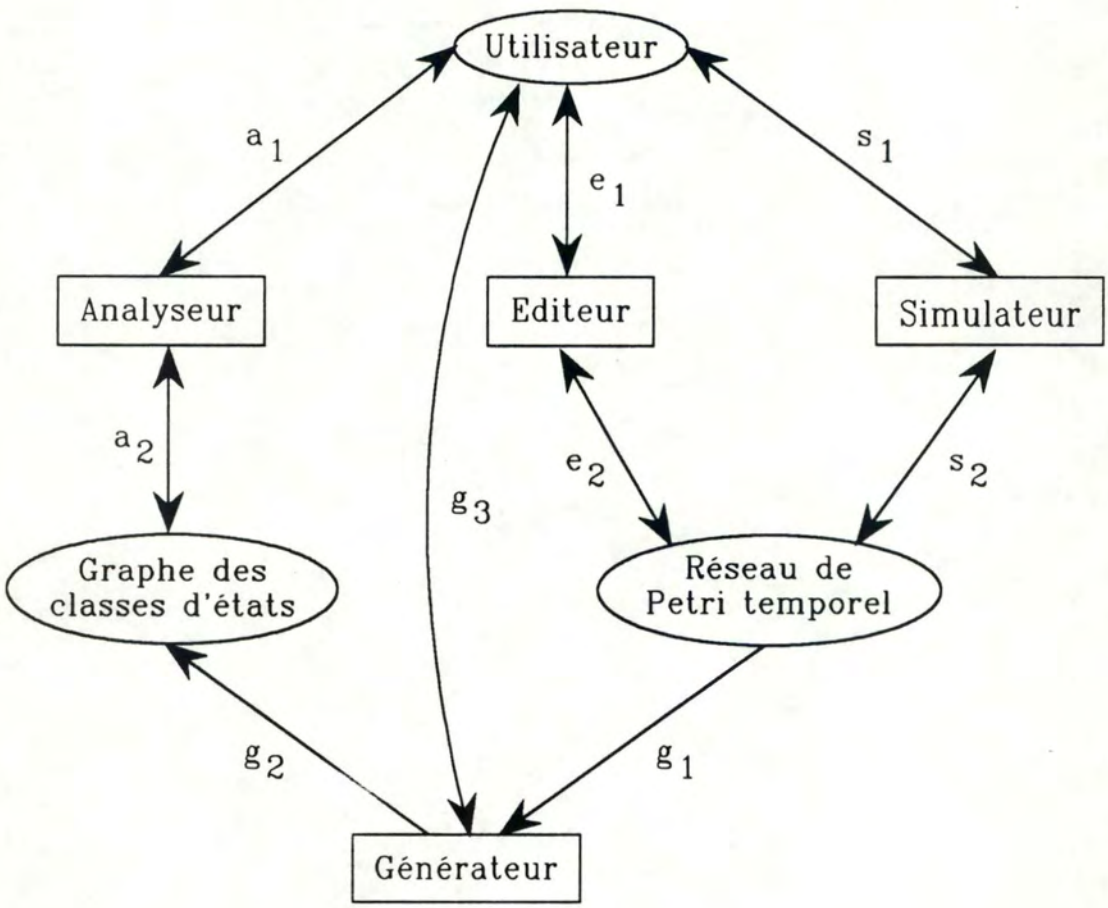


Figure 4.1. Architecture modulaire

4.2. Conception d'un outil pour l'analyse des réseaux de Petri temporels

Le logiciel d'analyse que nous allons développer dans cette section se présente sous la forme d'un outil interactif mono-utilisateur.

Pour l'implémentation de celui-ci, nous recourons au langage de programmation logique *Prolog* [CLO 87] [COE 88] [SHA 86].

Prolog offre en effet plusieurs avantages :

- *il est concis* : quelques clauses suffisent à définir des calculs relativement sophistiqués.
- *il permet l'indépendance par rapport aux problèmes de mise en oeuvre* : l'interpréteur Prolog contenant sa propre stratégie de résolution, il est possible de décrire un problème sans faire référence à une stratégie de contrôle ou à un algorithme particulier pour les enchaînements. Il est ainsi possible d'exprimer réellement le "quoi" sans se soucier du "comment".
- *il facilite une approche de type prototypage*, c'est-à-dire essentiellement qu'il conduit à des spécifications exécutables.

4.2.1. Architecture générale

L'outil présenté ici fournit à l'utilisateur les éléments nécessaires pour mener à bien la méthode d'analyse des réseaux de Petri temporels présentée dans les chapitres précédents.

Il se compose de quatre modules principaux dont l'organisation est représentée par la figure 4.1. :

1. L'*éditeur* qui permet l'entrée ou la modification d'un réseau de Petri temporel de manière simple (flèches e_1 et e_2).
2. Le *simulateur* qui permet la simulation du modèle opérationnel par l'exécution du modèle de réseau de Petri temporel correspondant (flèches s_1 et s_2).
   Comme l'exécution se fait pas à pas, par le tir d'une seule transition à la fois et à la demande de l'utilisateur, il doit y avoir un haut degré d'interaction avec l'utilisateur.
3. Le *générateur du graphe des classes d'états* qui génère le graphe représentatif du comportement du système modélisé (flèche g_2) à partir du réseau de Petri temporel (flèche g_1). Notons dès à présent que la flèche g_3 provient du fait que si la condition suffisante pour que le réseau soit borné n'est pas respectée, le générateur en avertit l'utilisateur; libre alors à celui-ci de continuer ou de stopper.
4. L'*analyseur* qui, outre la possibilité de visualiser le graphe des classes d'états construit par le générateur, offre à l'utilisateur (flèche a_1) un ensemble de fonctions permettant l'étude (flèche a_2) du graphe.

Dans les paragraphes qui vont suivre nous allons reprendre plus en détail chacun de ces modules.

4.2.2. L'édition des réseaux de Petri

La description du réseau de Petri temporel est préalablement nécessaire à l'exécution du simulateur et du générateur du graphe des classes d'états.

Dans une première approche, nous avons utilisé l'éditeur de texte Emacs. Mais, l'inconvénient majeur de cette solution est qu'un tel éditeur n'offre pas de contrôleur de syntaxe qui, de manière interactive, avertit l'utilisateur lorsqu'un élément du réseau introduit ne respecte pas la syntaxe d'un réseau temporel; par exemple, lorsque l'utilisateur introduit des bornes temporelles négatives.

Principes de représentation

Les réseaux de Petri temporels sont représentés sous la forme d'un ensemble d'assertions logiques (assertions Prolog) dont la syntaxe est semblable à celle utilisée dans le logiciel PIPN[1] [PIP 88].

Les quatre éléments importants pour la spécification d'un réseau de Petri temporel sont l'ensemble des places, l'état initial, l'ensemble des transitions et l'ensemble des définitions des conditions sur les données (préconditions, postconditions, intervalles de tir).

1° La déclaration des places

Nous allons coder l'état initial comme une assertion ayant la même forme qu'une liste de prédicats.

Par exemple, on code les places de la figure 2.1. par l'assertion Prolog :

place([p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8]).

Syntaxe :

< DéclarationPLaces > :== place([{ < ListePlaces > }]).
< ListePlaces > :== < Terme > {, < ListePlaces > }.

1 PIPN est un outil offrant un environnement de prototypage pour les modèles écrits dans le formalisme des réseaux de Petri à prédicats. Pour davantage d'informations concernant cet outil, le lecteur est renvoyé à [HAS 89].

Contraintes :

Les places sont des termes quelconques, mais deux places distinctes ne doivent pas être unifiables; toute précondition ou postcondition doit être l'instance d'une place unique.

2° La spécification de l'état initial

L'état initial du réseau de Petri temporel représenté à la figure 2.1. est codé par l'assertion Prolog :

init([p_1]).

Syntaxe :

<SpécificationEtatInitial> ::= init([{ <ListePropositions> }]).
<ListePropositions> ::= <InstancePlaceInit> {, <ListePropositions> }.
<InstancePlaceInit> ::= <Terme>.

Contraintes :

L'état initial est un ensemble de propositions, ce sont les propositions vraies dans un état initial du système.

Ces propositions sont encore des instances de base des places déclarées.

3° La déclaration des transitions

Une transition est définie par un identificateur, un ensemble de préconditions et de postconditions, et un intervalle de tir.

Syntaxe :

<SpécificationTransition> ::= tr({ <IdentifTransition> },
pré([{ <Preconditions> }]),
post([{ <Postconditions> }]),
delay(<IntervalleTemps>)
).
<IdentifTransition> ::= <Atome>.
<Preconditions> ::= <InstancePlace> {, <Préconditions> }.
<Postconditions> ::= <InstancePlace> {, <Postconditions> }.
<InstancePlace> ::= <Terme>.
<IntervalleTemps> ::= <NombrePositif>, <NombrePositifInfini>.
<NombrePositif> ::= <Nbr $\in$ $Q^+ \cup 0$>.
<NombrePositifInfini> ::= <Nbr $\in$ $Q^+ \cup 0 \cup$ '*'>.
où Nbr désigne un nombre,
'*' désigne l'infini.

Contraintes :

L'identificateur de la transition ne peut être une variable et deux transitions ne peuvent avoir le même identificateur.

Une transition ne peut avoir deux préconditions ou deux postconditions identiques.

Les préconditions et postconditions sont des instances des places déclarées.

4.2.3. Le simulateur

L'expérience a montré que la simulation permet de rechercher et de localiser un nombre important d'erreurs d'intégrité et d'erreurs sémantiques du modèle opérationnel [PAP 86].

Par exemple, l'utilisateur peut d'abord examiner les scénarios[1] normaux d'un modèle opérationnel, qui en principe sont des cas simples, et ensuite passer à l'examen de situations anormales. De cette façon et pour autant que la taille du modèle le permette, l'utilisateur peut vérifier l'ensemble des comportements du système.

Toutefois, cette méthode de recherche d'erreurs dans le modèle opérationnel présente des limites. Ces limites concernent principalement la capacité de l'utilisateur à comprendre le comportement du modèle lorsque ce dernier atteint une taille difficilement manipulable par l'esprit humain.

La taille réduite des applications constitue donc un facteur important pour une simulation intelligible.

A) Principe d'implémentation

Le simulateur est construit à partir de la méthode par énumération des classes d'états d'un réseau de Petri temporel.

La règle de tir des transitions et le calcul des classes suivantes sont ceux présentés au chapitre 2. Le lecteur intéressé est donc renvoyé à ce chapitre. De même, il trouvera en annexe 1 le texte source du programme Prolog.

B) Fonctionnement du simulateur

Le simulateur permet de faire évoluer un réseau de Petri temporel pas à pas à partir de sa classe initiale $C_0 = (M_0, D_0)$ en proposant à l'utilisateur de tirer une transition parmi celles proposées à chaque pas de la simulation.

Il est activé à partir de l'environnement Prolog et par la commande simu.

---

1 Un scénario est vu ici comme un chemin comprenant certaines transitions.

Une fois le simulateur activé, l'utilisateur est invité à donner le nom du fichier dans lequel le réseau de Petri temporel a été décrit :

"Entrez le nom du fichier contenant la description du réseau de Petri temporel : "

Le simulateur donne alors l'état initial et la liste des transitions sensibilisées dans cet état, de même que les intervalles de tir associés à chaque transition ainsi que les relations d'interdépendance.

$$C_0 = M_0 : \ldots$$
$$D_0 : \ldots$$

L'utilsateur a alors la possibilité de tirer une transition ou encore de stopper l'exécution.

Dans le cas du choix d'une transition non sensibilisée, le simulateur renvoie un message d'erreur :

"Erreur ! Cette transition n'est pas sensibilisée."

et il repropose alors la liste des transitions sensibilisées.

Si la transition choisie peut être tirée, le simulateur calcule alors la nouvelle classe engendrée par le tir de la transition selon l'algorithme défini au chapitre 2. Il donne les éléments de la classe et les nouvelles transitions sensibilisées.

Si le réseau est borné, il est ainsi possible d'obtenir l'ensemble complet des scénarios d'évolution et de tracer le graphe des classes d'états.

4.2.4. Le générateur du graphe des classes d'états

A) Principes de fonctionnement

Le générateur a comme entrée le réseau de Petri temporel et, à partir de celui-ci, il construit de manière automatique le graphe des classes d'états.

La génération du graphe est basée sur l'exécution totale du réseau de Petri temporel. L'exécution, tout comme pour le simulateur, se fait en utilisant la règle de tir et l'algorithme du calcul de la classe suivante défini au chapitre 2. Il est donc intéressant, afin de ne pas être redondant, de considérer un module commun à la fois au simulateur et au générateur. Ce module mettra en oeuvre les différentes étapes du calcul de la classe suivante.

Pour que cet outil puisse effectivement générer le graphe, le réseau doit être borné.

La condition suffisante développée au chapitre 3 revêt donc ici toute son importance. Celle-ci doit être vérifiée à chaque pas de l'algorithme. Dans le cas où elle ne le serait pas, le générateur en avertit l'utilisateur; libre alors à celui-ci de stopper l'exécution ou de continuer la

construction du graphe en reprenant l'exécution où elle s'est arrêtée. Dans un cas comme dans l'autre, il est vivement souhaité que les résultats intermédiaires puissent être consultés par l'utilisateur pour une analyse éventuelle du comportement partiel du réseau.

Nous soulignons ici le fait que ce module, de par le caractère suffisant et non nécessaire de la propriété borné d'un réseau de Petri temporel, nécessite un haut degré d'interaction entre l'outil et l'utilisateur.

B) Génération du graphe des classes d'états

Le générateur donne :

- le contenu des classes d'états sous la forme :

<DéclarationClasses> ::= C_i = M_i : [{ <Places> }]
D_i : [<DomaineTir>].

où

<Places> ::= <Terme> {, <Places> }.
<DomaineTir> ::= { <Inéquations> } , { <Interdépendances> }.
{ <Inéquations> } ::= <NombrePositif> ≤ <Transition> ≤ <NombrePositifInfini> {, <Inéquations> }.
{ <Interdépendances> } ::= <Transition> _ <Transition'> ≤ <Nombre> {, <Interdépendances> }.
où <Transition> ≠ <Transition'>.
<Transition> ::= <Terme>.
<Transition'> ::= <Terme>.
<Nombre> ::= <Nbr ∈ Q ∪ '*'>.
<NombrePositif> ::= <Nbr ∈ Q^+ ∪ 0>.
<NombrePositifInfini> ::= <Nbr ∈ Q^+ ∪ 0 ∪ '*'>.
où Nbr désigne un nombre,
'*' désigne l'infini.

- le graphe des classes d'états constitué d'un ensemble de lignes ayant la forme :

$$C_k \rightarrow t_i \in [a_i, b_i] \setminus C_l$$

ce qui signifie que le tir de la transition t_i depuis la classe C_k à un instant compris entre a_i et b_i conduit à la classe C_l.

4.2.5. Analyseur du graphe des classes d'états

L'analyseur devrait pouvoir intégrer un ensemble de fonctions (fonctions utilitaires pour l'examen des chemins du graphe) permettant de déduire les propriétés du réseau à partir du graphe des classes d'états donné par le générateur et de sa connexité.

En effet, lorsqu'il est fini, le graphe des classes d'états est un instrument idéal pour l'analyse du comportement du système modélisé par le réseau de Petri temporel.

A partir de sa connexité, il est possible de prouver l'absence de blocage dans le réseau, de même que les propriétés vivant et réinitialisable du réseau.

Toute classe d'états du graphe dont le domaine de tir est vide (marquage bloquant) représente une situation de blocage dans le réseau.

La propriété vivant est complètement établie pour un réseau borné. A partir du graphe des classes d'états, il est possible de vérifier si des transitions ne sont jamais tirées et quelles sont ces transitions.

Le graphe des composantes fortement connexes maximales déduit du graphe des classes d'états, outre le fait qu'il permet de décrire les différentes phases de fonctionnement du système, contient des informations très utiles pour l'étude de la propriété réinitialisable du réseau.

Par définition, un réseau de Petri temporel est réinitialisable si sa classe initiale est réaccessible depuis n'importe quelle autre classe. D'un point de vue connexité, cela se traduit par l'existence d'une seule composante fortement connexe maximale. Notons au passage que la propriété réinitialisable est ici plus restrictive que dans le cas des réseaux de Petri classique. Il se peut très bien, qu'un réseau temporel, bien que vivant, soit non réinitialisable en raison des contraintes temporelles.

Ainsi qu'il a été constaté dans [ROU 85], les modèles à réseaux de Petri temporels de systèmes réels exécutent souvent une évolution transitoire avant d'atteindre leur régime permanent. Les intervalles de temps initiaux affectés aux transitions ne sont plus reproduits pendant le fonctionnement, parce que des relations d'interdépendance apparaissent entre les dates de tir de certaines transitions. De par sa définition, le graphe des composantes fortement connexes montre clairement les régimes transitoires et aide à en déduire les causes.

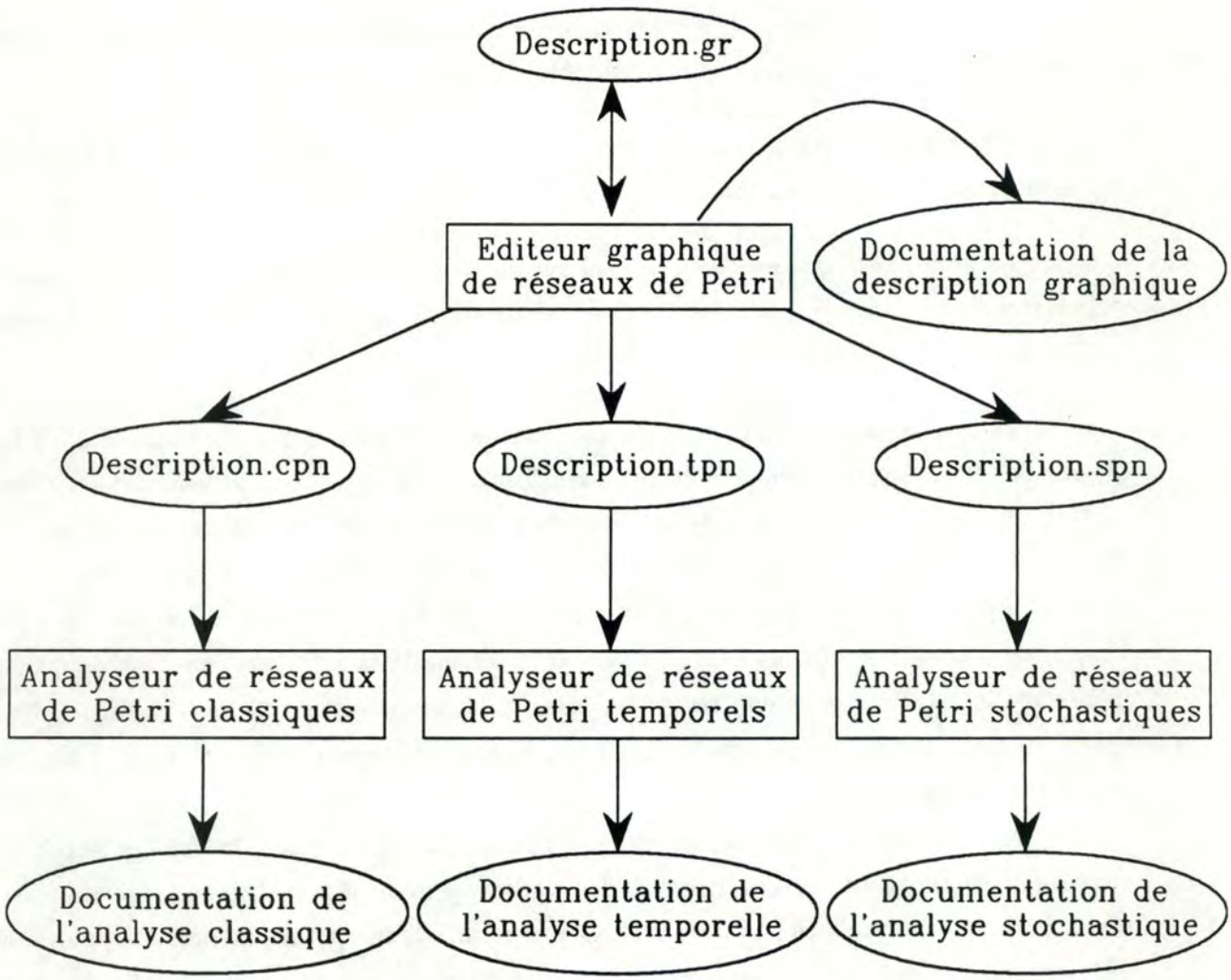


Figure 4.2. Schéma simplifié de l'architecture de l'outil RdP

4.3. RdP : outil d'aide à la conception et à l'évaluation de systèmes par réseaux de Petri

4.3.1. Présentation générale

RdP [RDP 89] est un outil d'aide pour la modélisation et l'analyse des systèmes au moyen du formalisme des Réseaux de Petri et qui permet une approche modulaire d'un problème.

Il peut manipuler trois classes de réseaux de Petri :

1. Les *réseaux de Petri classiques*, utilisés pour la validation qualitative des systèmes, indépendamment de toute notion de temps.
2. Les *réseaux de Petri temporels*, adaptés à la validation qualitative temporelle de systèmes dépendants du temps.
3. Les *réseaux de Petri stochastiques*, permettant essentiellement l'évaluation quantitative de performance de systèmes temps réel, où la durée de chaque action élémentaire est aléatoire.

Cet outil se compose de quatre modules principaux (Cf. figure 4.2.) :

1. L'*éditeur graphique* qui permet de construire de manière souple et modulaire un réseau de Petri, qu'il soit classique, temporel ou encore stochastique.
2. Un *analyseur de réseaux de Petri classiques*.
3. Un *analyseur de réseaux de Petri temporels*.
4. Un *analyseur de réseaux de Petri stochastiques*.

L'éditeur graphique travaille sur la forme interne graphique des réseaux de Petri tandis que les trois analyseurs utilisent des formes internes textuelles engendrées automatiquement par l'éditeur.

4.3.2. Fonctionnalités de RdP

A) L'éditeur

1° Principaux concepts

Les réseaux sont rentrés sous forme graphique en utilisant l'éditeur RdP. Nous avons trouvé ce dernier très convivial pour un utilisateur ayant un minimum de connaissance sur le formalisme des réseaux de Petri.

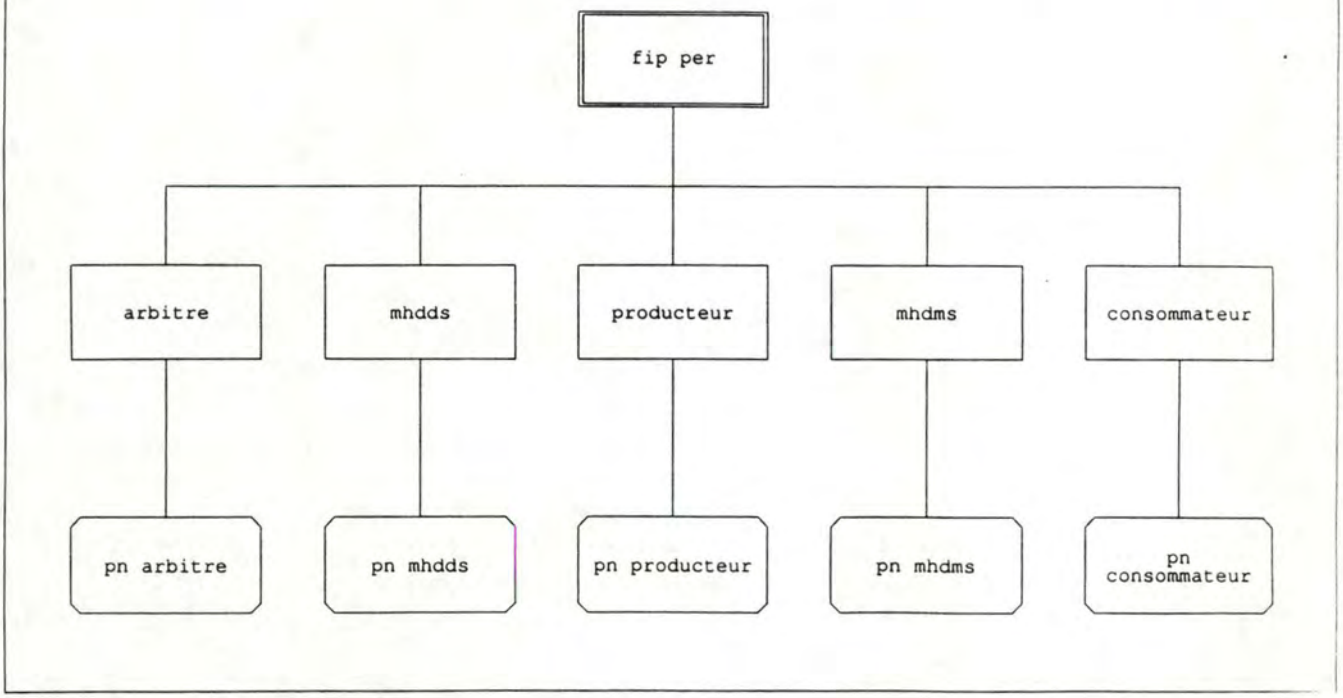


Figure 4.3.

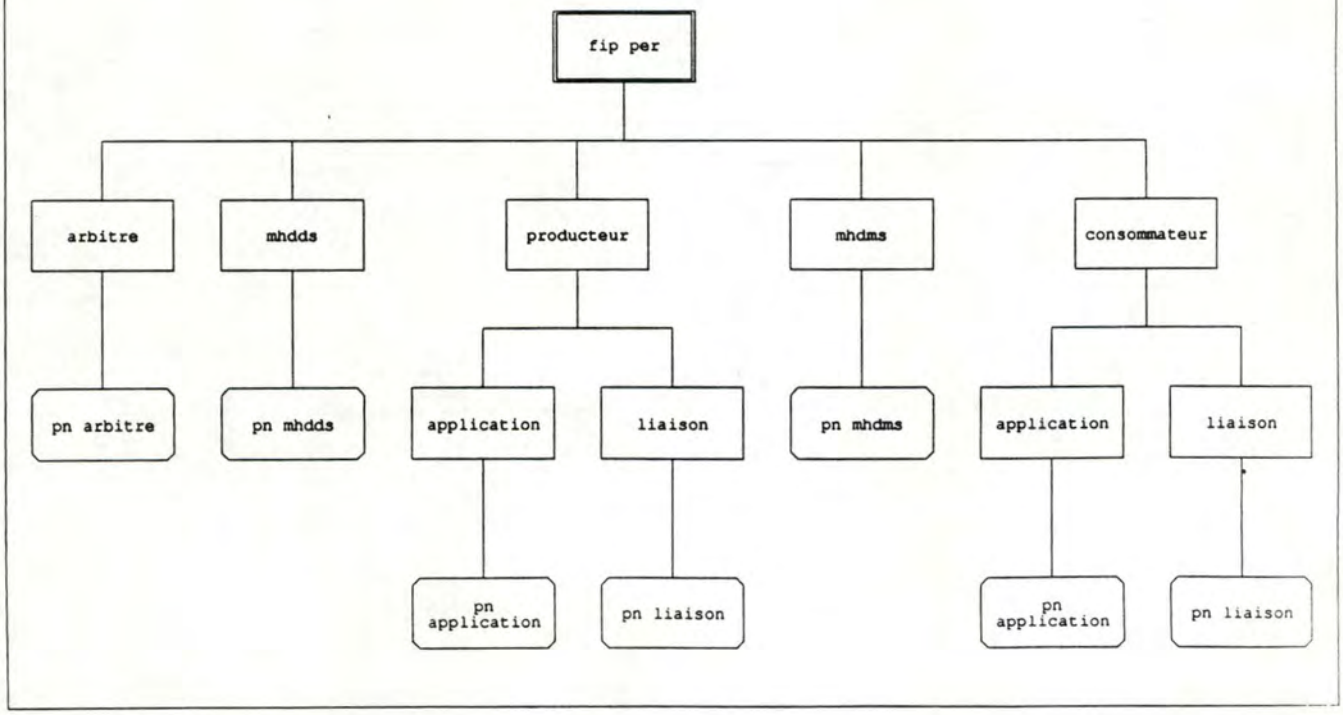


Figure 4.4.

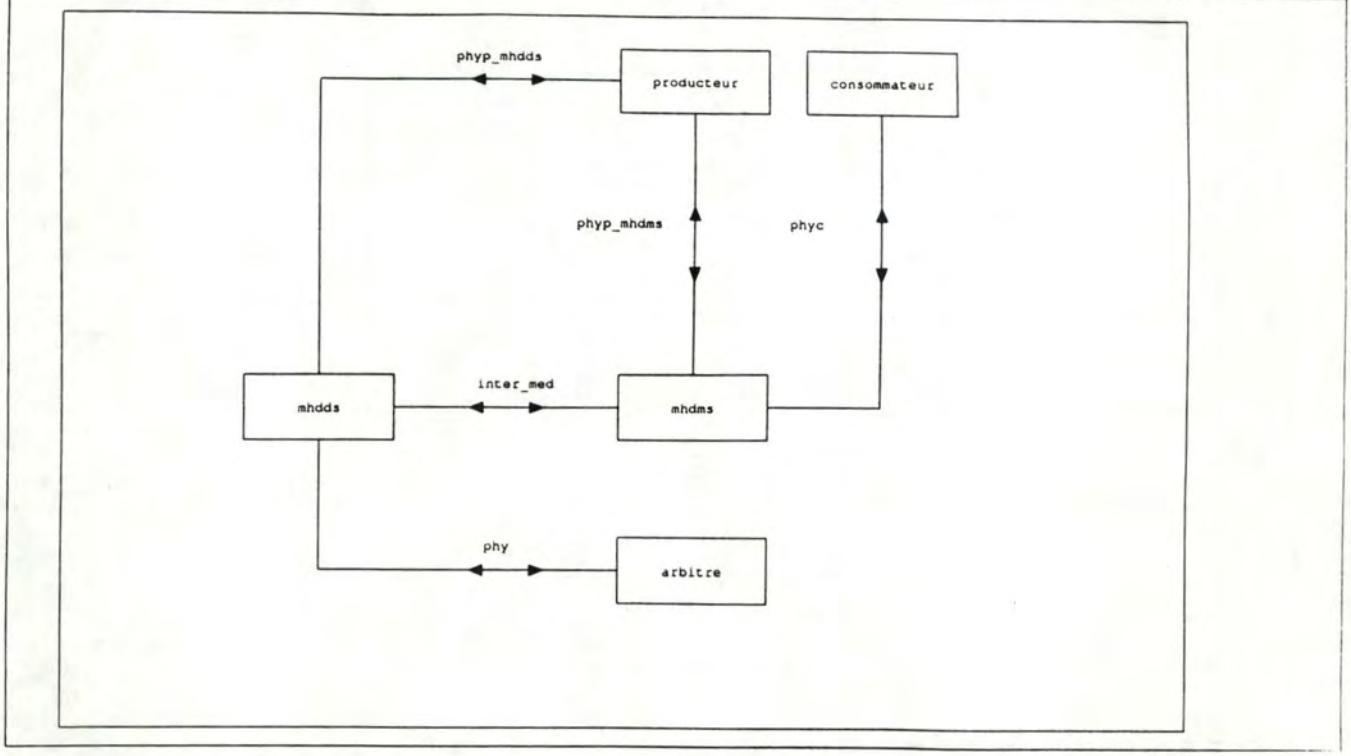


Figure 4.5.

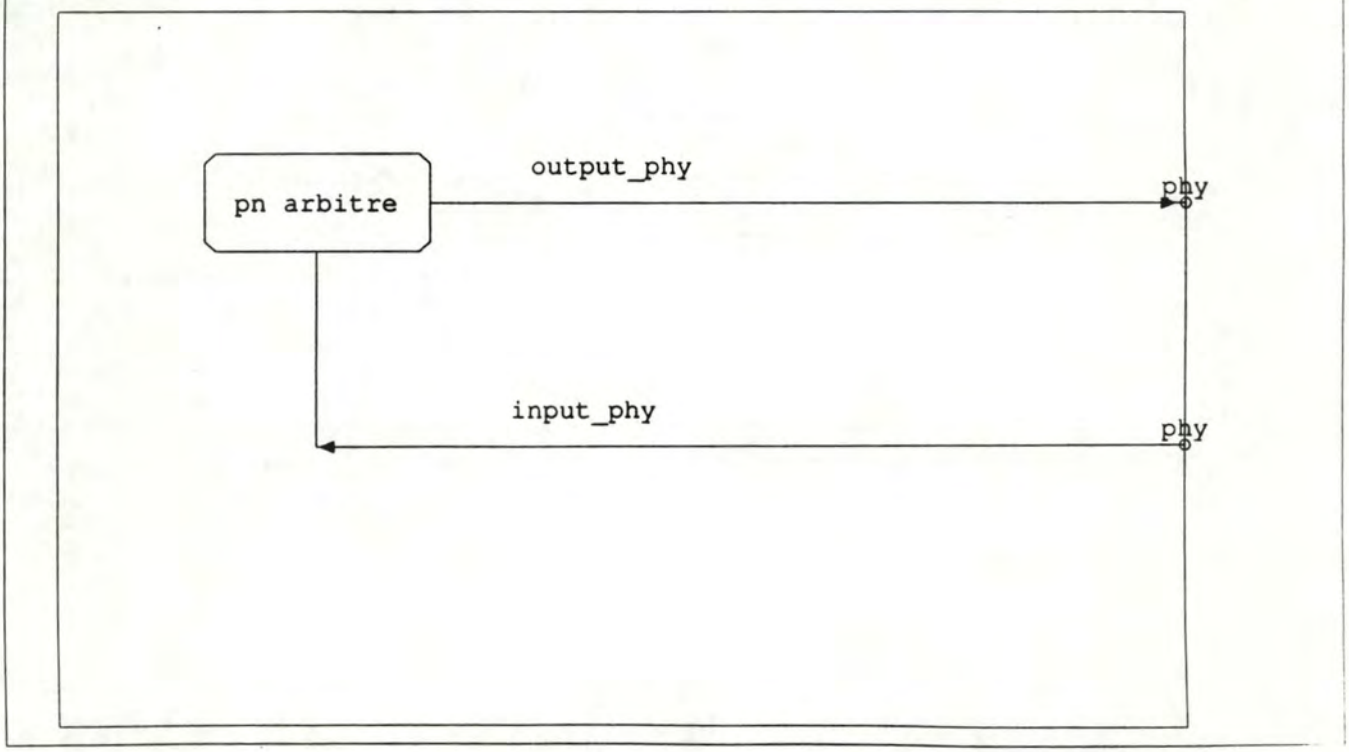


Figure 4.6.

L'éditeur permet :

- une édition graphique, orientée par la syntaxe, de l'architecture modulaire du réseau de Petri global et de la structure de ses divers modules : toutes les commandes proposées conduisent à un schéma syntaxiquement correct.
- un contrôle sémantique partiel sur les identificateurs et la représentation graphique sans entraîner de contrainte pour l'utilisateur (ex. : communications valides entre transitions ou places).
- un contrôle sémantique complet effectué à la demande de l'utilisateur pour localiser d'éventuelles erreurs.
- la création d'un fichier de description textuelle du modèle global.
- la création d'une documentation associée à la description du modèle saisi.

2° Méthode de description[1]

. La vue hiérarchique

La vue proposée par la figure 4.3. décrit le système *fip-per* (symbole ) décrit par réseaux de Petri.

Le système est vu comme une architecture de sous-systèmes communicants (les sous-systèmes *arbitre*, *mhdds*, *producteur*, *mhdms*, *consommateur*). A chaque sous-système est associé l'objet bloc (symbole ). Chaque bloc peut être lui-même raffiné en plusieurs autres blocs (Cf. figure 4.4.), ceci en fonction du niveau de raffinement désiré.

Au niveau le plus bas de cette décomposition en blocs et sous blocs, le comportement du sous-système représenté par un bloc peut être décrit par un réseau de Petri indépendant (symbole).

. La vue communication

Les communications entre sous-systèmes sont exprimées par des canaux (Cf. figure 4.5.). Ces canaux relient les blocs de même niveau ou de deux niveaux adjacents ou encore un réseau de Petri et son niveau bloc englobant directement supérieur.

Un canal reliant deux blocs peut être uni-directionnel ou bi-directionnel (sur la figure 4.5. le canal *phy* est bi-directionnel et relie le bloc *arbitre* au bloc *mhdds*).

La communication entre un réseau de Petri et son bloc englobant est obligatoirement uni-directionnel (Cf. figure 4.6. où le canal *output-phy* est déclaré sortant du réseau de Petri *pn arbitre* et se connecte au canal *phy* de niveau supérieur tandis que le canal *input_phy* est

---

1 Les figures reprises dans cette section sont tirées de la modélisation que nous avons faite du protocole liaison de données du réseau FIP, qui sera analysé au chapitre 6.
FIP est un système de communication adapté à la transmission d'informations entre capteurs, actionneurs et automates dans une installation industrielle automatisée.

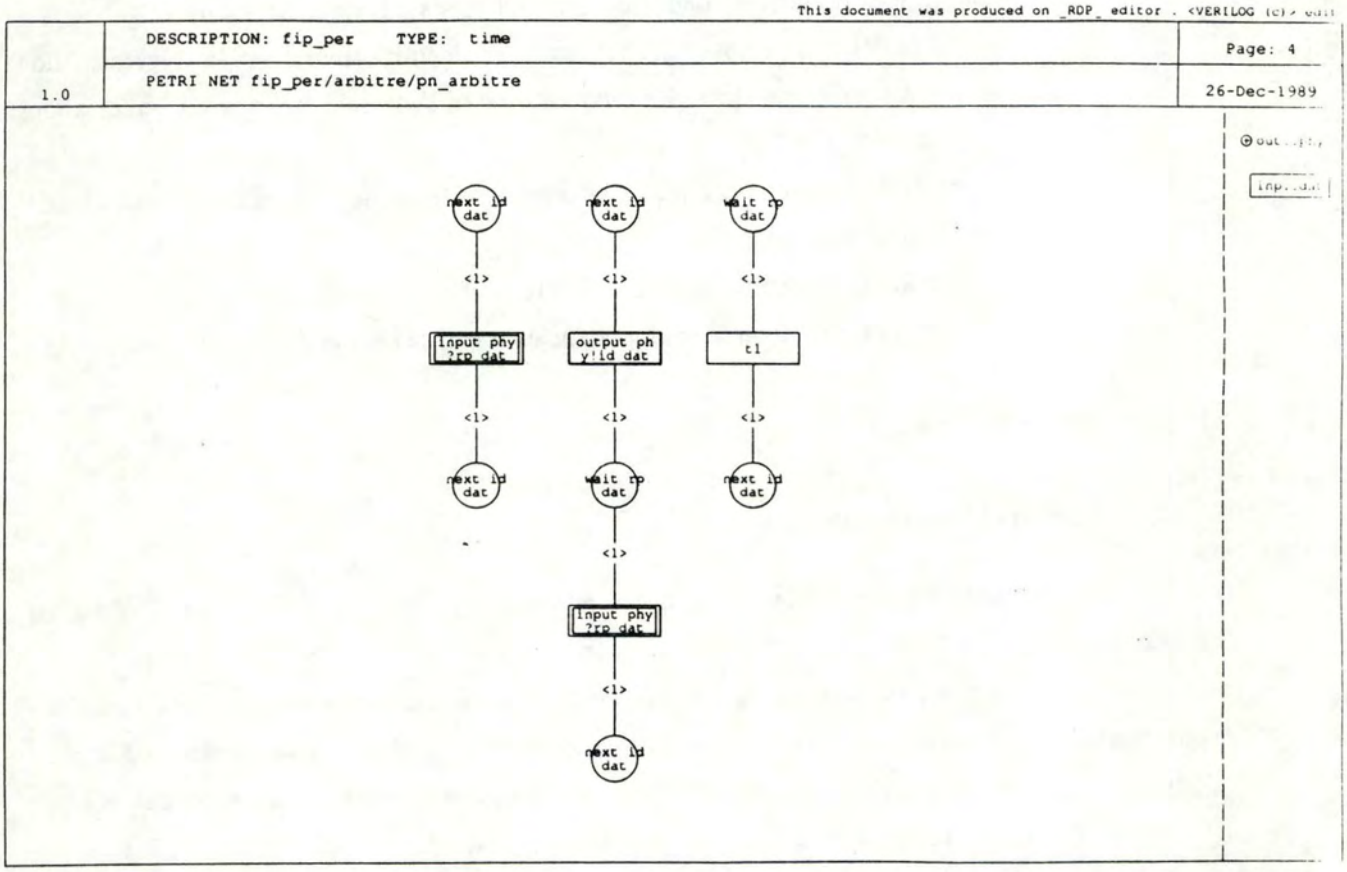

This document was produced on _RDP_ editor .
DESCRIPTION: fip_per TYPE: time
Page: 4
1.0
PETRI NET fip_per/arbitre/pn_arbitre
26-Dec-1989
next id dat
wait rp dat
<1>
input phy ?rp dat
output phy !id dat
t1

Figure 4.7.

décladé entrant dans le réseau de Petri *pn arbitre* et est également connecté au canal *phy* de niveau supérieur).

Au niveau des réseaux de Petri, ces canaux permettent la définition de *communications par rendez-vous*[1] (synchronisation de transitions) ou *par places partagées*[2]; il suffit d'indiquer le lien entre le canal faisant partie de l'environnement extérieur et la transition ou la place.

. La vue réseau de Petri

Le réseau présenté à la figure 4.7. possède deux places, *next_id_dat* et *wait_rp_dat*; les différentes représentations de la place *next_id_dat* (*wait_rp_dat*) représentent le même objet. Ce réseau possède également trois transitions, dont deux (*input phy?rp_dat* et *output phy!id_dat*) expriment des communications par rendez-vous, alors que la troisième (*t1*) n'exprime qu'un fonctionnement local.

L'interface avec l'environnement (à droite sur la figure 4.7.) comporte la liste des canaux sortant du réseau de Petri (*output_phy*), les transitions en input (*input phy?rp_dat*), les places importées (absentes ici).

B) L'analyseur

1° Différents analyseurs

Ainsi que nous l'avons déjà mentionné, RdP possède trois analyseurs. Avant de parler plus longuement de l'analyseur temporel, nous avons jugé utile, afin de souligner la puissance de l'outil, de présenter en quelques mots les possibilités offertes au niveau des réseaux classiques et stochastiques.

L'analyseur de réseaux de Petri classiques opère à partir du fichier de description textuelle du modèle global classique. Il permet à l'utilisateur de valider qualitativement le modèle global obtenu, grâce à :

- la construction du graphe des marquages accessibles,
- l'obtention des semi-flots du réseau,
- l'utilisation interactive de commandes permettant de visualiser la structure et les propriétés du réseau, de son graphe des marquages accessibles et de ses semi-flots,
- la création d'une documentation contenant les résultats qualitatifs de l'analyse.

---

1 Le mécanisme de rendez-vous consiste à fusionner les transitions communicantes de différents modèles en une transition commune.

2 Lorsque une ou plusieurs transitions de modèles distincts référencent une même place, cette place est dite partagée.

L'analyseur de réseaux de Petri stochastiques opère à partir du fichier de description textuelle du modèle global stochastique : il offre à l'utilisateur la possibilité d'effectuer à la fois une évaluation qualitative et quantitative des performances du système décrit et cela grâce à :

- l'obtention des divers critères quantitatifs de performance du système,
- la création d'une documentation contenant les résultats qualitatifs et quantitatifs de l'analyse.

2° L'analyseur de réseaux de Petri temporels

L'analyseur de réseaux de Petri temporels, RdPt, se présente sous la forme d'un outil interactif mono-utilisateur. Son environnement de travail propose un ensemble de fonctions, dites de haut niveau, qui permettent à l'utilisateur d'effectuer l'analyse.

L'indécidabilité de la propriété borné pour les réseaux de Petri temporels a engendré la nécessité de maintenir un haut degré d'interaction avec l'utilisateur afin que celui-ci puisse contrôler efficacement l'énumération.

. Préparation de la session d'énumération des classes d'états

Le lancement de la session d'énumération des classes d'états s'effectue à l'aide de la commande unix *rdpt*, suivi du nom du fichier d'étude.

L'analyseur donne alors les options de configuration par défaut (conditions spécifiques, incréments en temps CPU et en nombre de classes).

Avant le lancement proprement dit de l'énumération, l'utilisateur peut modifier ces options.

Les conditions spécifiques (propriétés sauf, k-borné pour le marquage des places,...) seront vérifiées incrémentalement pour chaque classe générée.

La fixation des incréments par l'utilisateur sur le temps CPU et le nombre de classes vont lui permettre d'interrompre l'énumération de manière répétitive à son gré.

Quant à la condition suffisante développée au chapitre 3, elle est implantée par défaut. Notons qu'elle s'est avérée satisfaisante pour la plupart des applications significatives traitées avec cet outil.

. Lancement de la session d'énumération des classes d'états

Dès le lancement de la session d'énumération des classes d'états, l'analyseur affiche le nom de l'étude en cours, puis exécute plusieurs phases préliminaires : réservation de la place mémoire, initialisation du modèle et lancement de l'analyse proprement dite.

Ensuite, le modèle incluant l'analyse syntaxique et sémantique du fichier descriptif est chargé. Si tout se passe correctement, un message le signale à l'utilisateur, avec le nom du modèle, ainsi que le nombre de places et de transitions qu'il contient.

L'analyse énumérative avec la construction du graphe des classes d'états peut alors se réaliser.

L'analyseur applique la règle de tir pour les classes d'états et calcule les classes suivantes en appliquant de manière répétitive l'algorithme présenté au chapitre 2.

Pour chaque transition tirable dans une classe, une nouvelle classe est générée et comparée avec ses prédécesseurs. Si elle n'existe pas encore, le graphe des classes d'états est complété.

Chaque fois qu'une classe d'états vient d'être construite, la validité des conditions spécifiques, y compris la condition suffisante, et le non dépassement des incréments sont vérifiés. Si une des conditions n'est plus respectée ou qu'un incrément en temps CPU ou en nombre de classes est atteint, la cause de l'anomalie est affichée et le contrôle de la session d'énumération est rendu à l'utilisateur.

L'utilisateur a alors le choix entre relancer la construction du graphe, la suspendre ou encore l'abandonner. Quel que soit l'état de l'analyse, l'utilisateur peut toujours interroger l'embryon du graphe déjà construit avant de redémarrer la construction. Si la construction du graphe des classes d'états s'achève de manière normale, l'analyseur affiche un message correct de l'analyse énumérative, avec le nombre des classes d'états obtenues et le temps CPU qui a été nécessaire à la construction du graphe; le contrôle est rendu à l'utilisateur qui peut alors commencer une interrogation interactive du graphe des classes d'états.

. Traitement du graphe des classes d'états

L'utilisation interactive d'un grand nombre de commandes permet à l'utilisateur de visualiser aussi bien la structure (marquage initial, visualisation des places, des transitions, des classes d'états, des composantes fortement connexes, ...) que les propriétés (vivant, réinitialisable, ...) du réseau et de son graphe de classes d'états. Ceci permet à l'utilisateur d'effectuer une validation qualitative temporelle du modèle global.

Finalement, une documentation complète des résultats qualitatifs de l'analyse peut être obtenue, ce qui, en milieu professionnel, offre un support papier non négligeable.

4.3.3. Critiques et perspectives de travail

Dans cette section, nous avons présenté les fonctionnalités d'un outil pour l'analyse des réseaux de Petri temporels.

Parmi les nombreux points positifs que nous avons relevés lors de son utilisation, nous retiendrons principalement le fait que RdP présente une interface graphique très agréable pour l'utilisateur et d'une manipulation aisée.

L'architecture sous forme de systèmes et sous-systèmes communicants nous a semblé particulièrement bien adaptée à la modélisation des protocoles de communication. Il est en effet possible de modéliser ainsi chaque entité d'un système séparément et de représenter facilement les points d'interaction entre ces entités.

RdP, de par l'ensemble des fonctions qu'il offre, permet également une analyse rapide et efficace du comportement et des propriétés du réseau modélisé.

La qualité de la documentation qu'il produit de manière automatique nous a également séduit.

A côté de nombreuses qualités, nous regrettons toutefois l'absence d'un simulateur qui permettrait de voir, pas à pas et de façon interactive, les évolutions possibles du système modélisé. Notons toutefois, que la commande *find firing sequence*[1] permet à cet effet, de visualiser un scénario.

Un autre problème que nous avons rencontré lors de la réalisation d'applications de taille plus importante, et qui est plus une caractéristique due au modèle des réseaux de Petri temporels qu'à l'outil lui-même, est l'explosion combinatoire du nombre de classes d'états. Dans de nombreux cas, les systèmes à vérifier sont trop complexes pour que l'on puisse construire leurs ensembles de comportements et les analyser. De façon pernicieuse, ce phénomène d'explosion peut conduire le concepteur à simplifier exagérément son modèle de telle sorte que celui-ci n'ait plus qu'une lointaine parenté avec le modèle initial.

Il serait donc utile d'envisager des techniques de réduction du nombre de classes d'états et d'apporter une attention particulière à la méthodologie de vérification pour que le nombre de classes d'états soit maintenu à une valeur raisonnable.

C'est l'absence de ces techniques de réduction que nous reprochons à l'outil RdP.

Nous pensons plus particulièrement aux techniques de projection développées au LAAS dans le cadre des réseaux classiques et du logiciel PIPN, ou encore à la définition de relations d'équivalence.

Dans le domaine des réseaux de Petri classiques nous avons eu l'occasion, durant notre stage à Toulouse, d'utiliser le logiciel PIPN et un complémentaire à celui-ci, NEC, qui permet de faire une vérification en précisant les événements (transitions) et/ou les propriétés (places) que l'on veut observer [VER 89]. On peut résumer les étapes de la vérification de la manière suivante :

1. Poser dès le départ ce que l'on veut vérifier et formuler le service attendu.

---

1 La fonction de cette commande est de donner une séquence de transitions franchissables dont le franchissement conduit d'une classe d'états donnée à une autre classe d'état donnée et contenant une transition donnée.

2. Sélectionner les agents (modules), les événements et les propriétés observables nécessaires pour effectuer la vérification.
   L'ensemble des événements associés aux transitions du réseau peut ainsi être divisé en deux sous-ensembles :
   . d'une part, le sous-ensemble des événements internes, c'est-à-dire ceux qui ne doivent plus apparaître dans la vue abstraite,
   . et d'autre part, le sous-ensemble des événements visibles qui doivent eux, continuer à apparaître dans la vue abstraite.
   L'ensemble des propriétés peut également être divisé de façon similaire.
3. Effectuer la projection.
4. Interpréter cette projection en comparant le modèle du service obtenu avec celui du service attendu.
5. Si la projection dévoile certaines indications, vérifier celles-ci en ajoutant à la projection des propriétés et/ou des événements observables.

Nous avons pu apprécier les services offerts par NEC, lors d'une application concernant l'introduction, au niveau de la couche liaison de données du protocole FIP, d'un mécanisme permettant de spécifier des accès simultanés en écriture et en lecture sur un registre.

Toutes ces méthodes permettent de réduire considérablement l'automate.

Concernant les méthodologies de vérification, nous ne saurions que trop recommander les travaux développés dans [VER 89].

Il serait donc intéressant de compléter l'outil RdP par des techniques similaires mais appliquées aux réseaux temporels.

4.4. Conclusion

Les outils que nous avons présentés dans ce chapitre ont été utilisés pour modéliser les applications figurant dans ce mémoire.

Nous avons tenu à expliciter les fonctionnalités d'un outil tel que RdP, sans toutefois entrer dans trop de détails, parce que nous sommes conscient que l'existence d'outils comme celui-là est une condition nécessaire à l'utilisation de la technique des réseaux de Petri dans le monde industriel. Il est donc important de faire connaître ces outils à travers leur mode d'utilisation sans toutefois perdre de vue leurs limites.

CHAPITRE 5

METHODOLOGIE D'ANALYSE ET UTILITE DES RESEAUX DE PETRI TEMPORELS

5.1. Introduction

L'objectif de ce chapitre est de montrer l'utilité des réseaux de Petri temporels pour l'analyse du fonctionnement de systèmes parallèles asynchrones dans lesquels le temps apparaît comme un paramètre.

Les protocoles de communication sont des exemples de tels systèmes : ils font un large usage de contraintes temporelles dans leur spécification. Par exemple, les mécanismes de reconfiguration après la perte de messages ou un changement de topologie du réseau sont habituellement implantés à l'aide de temporisations.

Les réseaux de Petri temporels et la méthode d'analyse par énumération des classes d'états permettent de vérifier que les valeurs de ces temporisations sont correctement choisies.

Dans une première partie, nous donnons différentes étapes à suivre pour la validation de protocoles.

Ensuite, nous montrons, à travers des exemples concrets, l'utilité des réseaux de Petri temporels et de la méthode d'analyse par énumération des classes d'états.

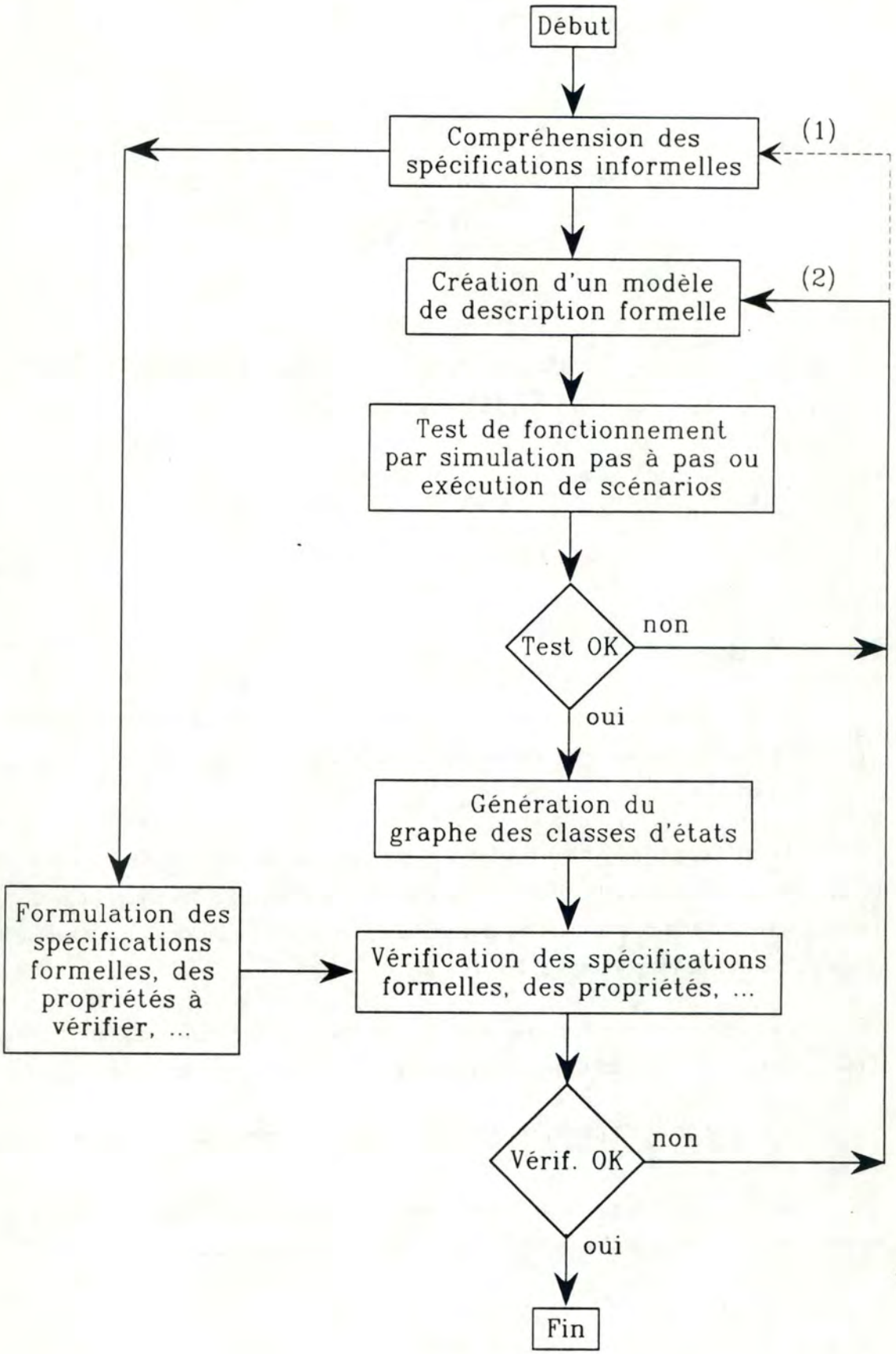


Figure 5.1. Méthodologie d'analyse

5.2. Méthodologie d'analyse

La complexité des systèmes distribués, complexité inhérente à la distribution géographique des fonctions et des supports, nécessite, lors de la conception, le suivi d'une méthodologie d'approche adaptée, partant des spécifications et arrivant aux tests [PAP 86].

C'est pourquoi, avant de présenter certaines applications des réseaux de Petri temporels, nous avons jugé utile de donner une méthodologie pour l'analyse de ces systèmes. Celle-ci s'avère particulièrement utile pour l'analyse de systèmes complexes, telle que celle que nous avons menée pour la couche liaison de données du protocole FIP et qui sera présentée au chapitre suivant. La méthode que nous proposons est fondée sur le bon sens et se présente sous la forme de cinq étapes :

- compréhension des spécifications informelles,
- création du modèle de description formelle sous forme d'un réseau de Petri temporel,
- test de fonctionnement du modèle de description formelle à l'aide d'un simulateur,
- génération du graphe des classes d'états équivalent au modèle de la description,
- vérification des propriétés du modèle et analyse de son comportement.

La figure 5.1. montre clairement l'enchaînement de ces étapes.

La flèche (1) représente une particularité liée à la complexité de conception de certains systèmes. En effet, l'utilisateur dispose initialement de spécifications informelles qui portent sur le comportement de l'objet sous modélisation. Il propose un modèle de description, en utilisant un formalisme fonctionnel, comme les réseaux de Petri temporels. Cette proposition est intuitive et tente de satisfaire les spécifications. Si ce n'est pas le cas, l'utilisateur peut réétudier la cohérence des spécifications (flèche (1)), ou bien reconstruire un autre modèle de description (flèche (2)). Ce choix doit être guidé par sa seule expérience.

5.2.1. Compréhension des spécifications informelles

Avant toute chose, il est nécessaire de bien saisir quelle est la nature exacte des spécifications car elles sont souvent ambigües et parfois même contradictoires.

5.2.2. Création du modèle de description formelle sous forme d'un réseau de Petri temporel

La méthodologie de modélisation actuellement utilisée dans les études sur les systèmes distribués et les protocoles repose sur une représentation par Réseaux de Petri des tâches, proche

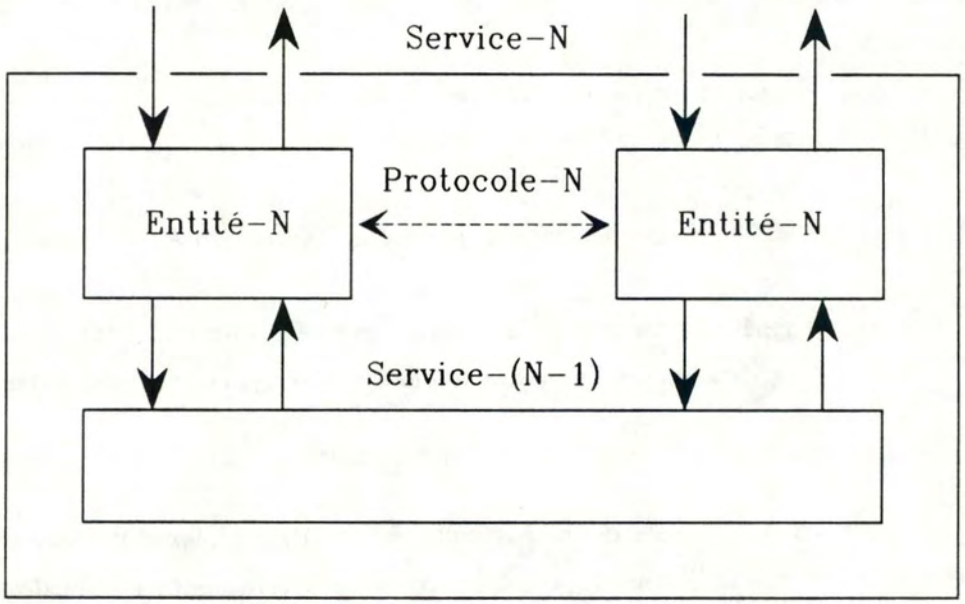


Figure 5.2. Architecture en couches

du concept de couche du modèle de référence de l'ISO [AYA 85b]. Ce modèle de référence décrit l'architecture d'un système distribué comme une architecture hiérarchisée [ZIM 80] formée de plusieurs couches indépendantes de protocoles.

Chaque couche de protocole est constituée d'un ensemble d'entités coopérant selon un ensemble de règles, le protocole, en utilisant le service fourni par la couche immédiatement inférieure et fournissant elle-même un service à la couche immédiatement supérieure.

Dans le contexte d'une architecture multi-couches, l'environnement d'une entité est donc constitué par les deux niveaux qui lui sont adjacents et par les entités distantes.

Dès lors, la spécification de toute couche de protocole devrait inclure :

- la spécification du service de référence fourni par la couche de protocole,
- la spécification des entités communicantes de protocole,
- la spécification du service fourni par la couche inférieure et utilisé par la couche considérée.

Ainsi, la spécification d'un protocole décrit le comportement de chaque entité en réponse aux requêtes de service venant des entités de la couche supérieure et en réponse aux indications de service venant des entités de la couche inférieure.

Le fonctionnement global du protocole peut être obtenu en interconnectant les modèles de chaque entité locale au modèle qui décrit le comportement global du service de niveau inférieur tel qu'il est vu par les entités locales (Cf. figure 5.2.). On obtient ainsi le modèle global du protocole étudié qui tient compte à la fois des contraintes locales et globales du protocole.

En d'autres mots, cela revient à modéliser le comportement local de chaque entité et les interfaces reliant ces éléments.

A) Modèle local d'une entité de protocole

Dans une première étape, le comportement local de chaque entité de protocole est spécifié par un réseau de Petri, en l'occurrence ici, par un réseau de Petri temporel.

Si l'entité est complexe à modéliser, le modèle pour cette entité peut être conçu de manière modulaire et donc constitué de plusieurs activités ou sous-modèles.

Afin de traduire explicitement l'échange de messages, d'une part, entre les différents sous-modèles et, d'autre part, avec l'environnement, c'est-à-dire les couches de niveau inférieur et supérieur, nous étendons le formalisme des réseaux de Petri temporels à ce que nous appellerons les réseaux de Petri temporels étiquetés.

Les réseaux de Petri temporels étiquetés se distinguent des réseaux de Petri temporels, par le fait qu'on associe à chaque transition, en plus du nom et de l'intervalle de tir, une étiquette de la forme ?X/!Y. ?X/!Y est la notation de Hoare [HOA 78] pour représenter la réception du message X (notée ?X) et l'émission du message Y (notée !Y). Notons que les étiquettes ne sont pas nécessairement de la forme ?X/!Y, mais qu'elles peuvent aussi représenter seulement une réception (?X) ou seulement une émission (!Y).

L'interconnexion entre les différents sous-modèles peut être obtenue en fusionnant toute paire de transitions dont l'une est étiquetée par ?X et l'autre par !X.

Le modèle résultant de l'interconnexion des différents sous-modèles représente le modèle complet de l'entité dans lequel toutes les transitions étiquetées concernent les interactions avec l'environnement.

B) Modélisation du service utilisé

La spécification du service fourni par la couche adjacente inférieure peut être plus ou moins complexe. Ceci est dû au fait que le service a la possibilté ou non d'agir sur les interactions entre entités homologues et d'engendrer lui-même des interactions [AYA 85b].

Une première approche dans la modélisation du service utilisé consiste à spécifier seulement le transfert de données où chaque interaction est simplement le résultat d'une primitive d'envoi d'un message. Une telle modélisation est appelée *medium virtuel*.

On peut également associer différents comportements à ce medium virtuel : medium parfait, perte de messages, duplications de messages, ordonnancement FIFO, ...

Une deuxième approche est donnée par la modélisation d'un *service orienté connexion*. La modélisation d'un service orienté connexion, c'est-à-dire où la connexion est explicite, implique la prise en compte dans le modèle des interactions associées à l'ouverture et à la fermeture de connexion. Notons que la fermeture de connexion ne résulte pas nécessairement d'une interaction engendrée par l'entité homologue distante, mais le service peut l'engendrer sur sa propre initiative [ALG 84a].

C) Modèle global de la couche de protocole

Le modèle global de la couche de protocole est obtenu en interconnectant le modèle de chaque entité locale avec le modèle du service fourni par la couche inférieure.

Le modèle de chaque entité locale a été spécifié au moyen d'un réseau de Petri temporel étiqueté dans lequel les transitions étiquetées représentent, ainsi que nous l'avons déjà dit, les interactions externes avec la couche inférieure ou supérieure. De même, le modèle représentant le service de la couche inférieure a été représenté par un réseau de Petri temporel étiqueté où les

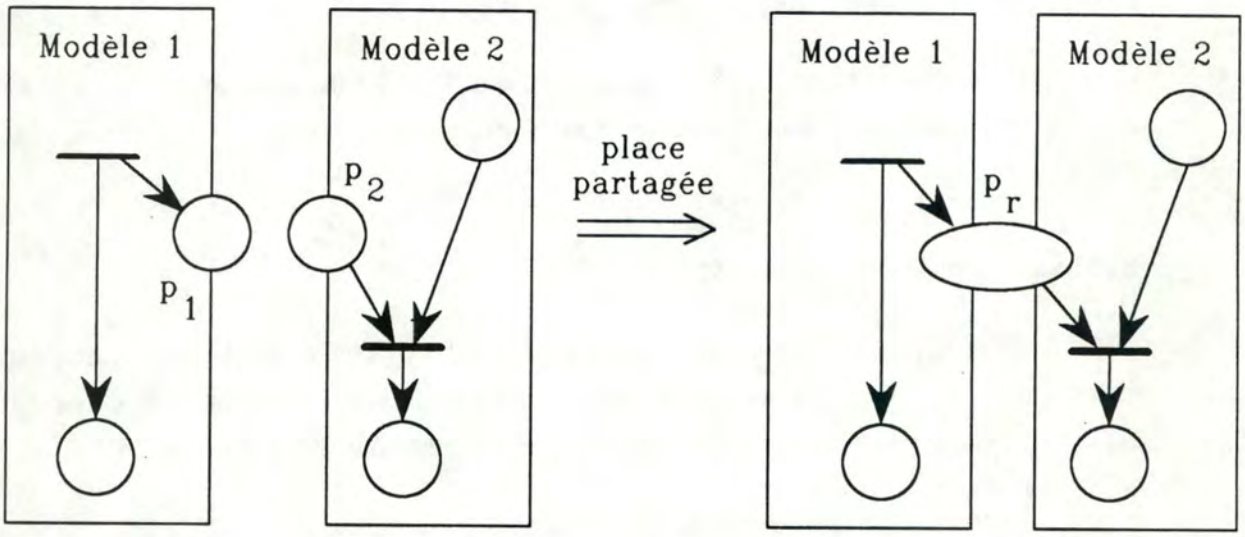


Figure 5.3. Communication par places partagées

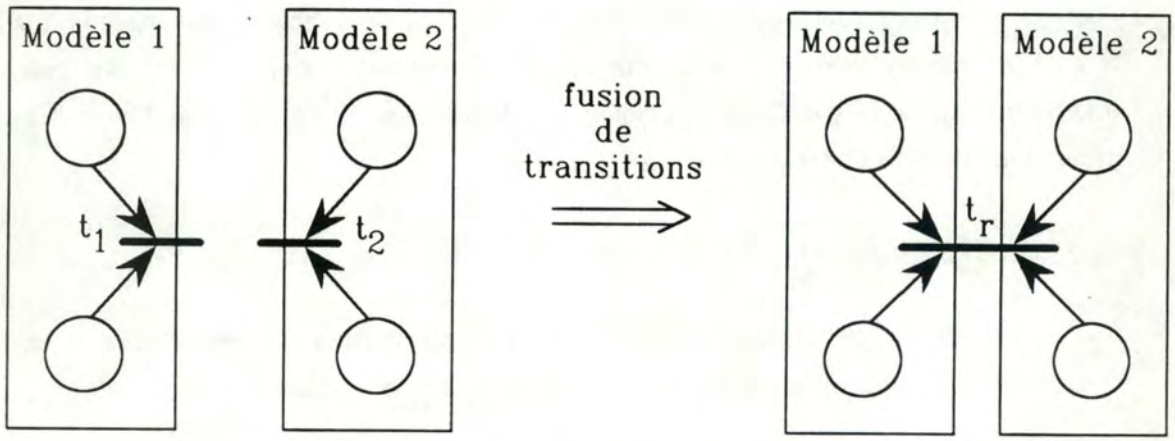


Figure 5.4. Communication par rendez-vous

étiquettes associées aux transitions représentent les interactions avec le modèle des différentes entités locales.

Dès lors, l'interconnexion de ces différents modèles donne également un réseau de Petri temporel étiqueté dans lequel les étiquettes représentent les interactions externes avec la couche supérieure.

Parmi les différentes manière de représenter l'interconnexion entre les modèles, nous retiendrons deux mécanismes :

- le mécanisme de *communication par places partagées*,
- le mécanisme de *communication par rendez-vous*.

. Communication par places partagées

La communication par places partagées représente un échange asynchrone où l'émetteur du message, lors de l'émission de celui-ci, ne se met pas en attente mais poursuit son exécution, tandis que le récepteur, s'il essaie d'accéder au message avant que celui-ci ne soit arrivé, est mis en attente.

Ce mécanisme est décrit dans chaque modèle par la création d'une place ayant la même sémantique dans chaque entité. L'interconnexion entre les différents modèles est alors réalisée en confondant ces places, ce qui permet d'obtenir une place commune aux entités, ainsi que le montre la figure 5.3.

. Communication par rendez-vous

La communication par rendez-vous représente un échange synchrone où l'entité émettrice tout comme l'entité destinatrice doit suspendre son exécution jusqu'à la réalisation de l'échange. L'entité qui envoie le message, ne le fera effectivement, que lorsqu'elle aura la garantie que le destinataire est prêt à le recevoir.

Le mécanisme de rendez-vous s'interprète de façon simple par la fusion de transitions (Cf. figure 5.4.). La transition résultante est constituée de la réunion des préconditions et de la réunion des postconditions des transitions d'origine. Son étiquette résulte du produit des étiquettes d'origine.

5.2.3. Test de fonctionnement du modèle de description formelle

Le fonctionnement du modèle de description formelle est testé à l'aide d'un environnement de simulation qui permet, d'une part, de faire évoluer le modèle à partir de son état initial

en sélectionnant une transition parmi celles proposées à chaque pas de la simulation et, d'autre part, d'exécuter des scénarios d'évolution.

La simulation ne permet pas de prouver qu'un système est correct, mais elle peut être très précieuse pour détecter des erreurs pendant la conception. De plus, par l'exécution de scénarios, elle permet une meilleure compréhension de systèmes compliqués et peut même être utilisée à des fins de démonstration.

5.2.4. Génération du graphe des classes d'états

Le graphe des classes d'états représente l'ensemble des comportements du système modélisé. Une analyse approfondie de ce graphe va permettre de valider ou non le modèle.

Au point suivant, nous allons montrer comment il est possible d'utiliser les résultats classiques sur la validation des réseaux de Petri pour analyser un protocole donné.

5.2.5. Vérification des propriétés du modèle et analyse de son comportement

Tout comme dans le cas de réseaux de Petri classiques, l'analyse de protocoles à partir des réseaux de Petri temporels permet de prouver deux classes de propriétés [AYA 85b] :

- des *propriétés générales* qui doivent être vraies pour tout protocole,
- des *propriétés spécifiques* qui sont propres au protocole considéré et qui dépendent de la sémantique associée aux places et aux transitions du réseau.

A) Propriétés générales

La première propriété que l'on peut souhaiter est que le protocole soit borné car si ce n'était pas le cas, cela impliquerait qu'il a un nombre infini d'états.

Une deuxième propriété que l'on peut vouloir montrer est que le protocole est sans blocage. Un protocole est dans une situation de blocage lorsqu'aucune transmission n'est possible dans l'état courant et qu'aucun message ne circule sur le bus interconnectant les entités.

Une troisième propriété que l'on peut désirer est que le modèle global du protocole soit réinitialisable, ce qui correspond bien au fonctionnement souvent répétitif d'un protocole. Mais, soulignons ici le fait que les modèles à réseaux de Petri temporels de systèmes réels, ainsi que nous l'avons déjà dit, exécutent souvent une exécution transitoire avant d'atteindre leur régime permanent. La configuration initiale des intervalles de temps affectés aux transitions n'est plus reproduite pendant le fonctionnement à cause de dépendances apparaissant entre les dates de tir de

certaines transitions. Il se peut donc que la classe initiale ne soit pas réaccessible sans pour autant traduire un mauvais fonctionnement du réseau.

On peut également vouloir vérifier qu'il n'y a pas de réception non spécifiée. En effet, si le bus n'altère pas l'ordre des messages qu'il transmet (Cf. file FIFO), les réceptions non spécifiées peuvent être caractérisées par une situation de blocage, où cependant certains messages sont en transit sur le bus. C'est le cas lorsque le message en tête de file ne peut être reçu par une entité réceptrice et que les messages suivant en transit sur le bus ne peuvent doubler le premier.

Quant à la propriété vivant, nous ne l'envisageons pas ici, car pour la vérification de temps-limites, par exemple, il est nécessaire de modéliser des situations de mauvais fonctionnement afin de pouvoir vérifier que les valeurs attribuées à ces temps-limites ne conduisent pas à ces situations. Dès lors, dans le cas d'un fonctionnement correct du système, les transitions modélisant ces situations d'erreurs, ne seront jamais tirées, et le réseau sera non vivant.

B) Propriétés spécifiques

Une propriété d'un protocole est dite spécifique lorsquelle dépend de la sémantique que le concepteur du protocole associe aux différents éléments du modèle global de ce protocole, c'est-à-dire les places, les transitions, ...

Dans les domaines des protocoles, on distingue fréquemment les propriétés liées à la correction partielle du protocole de celles liées à la progression du protocole.

Dans le cas de la correction partielle, on montre que si le protocole progresse alors il rend le service attendu. Cela consiste à vérifier l'adéquation entre la spécification du service que le protocole est censé rendre et le service réellement fourni par le protocole conçu.

Dans le second cas, on montre que le protocole progresse effectivement. La progression du protocole peut être prouvée en utilisant le graphe des classes d'états et la propriété typique à prouver est celle d'accessibilité par rapport à certains événements.

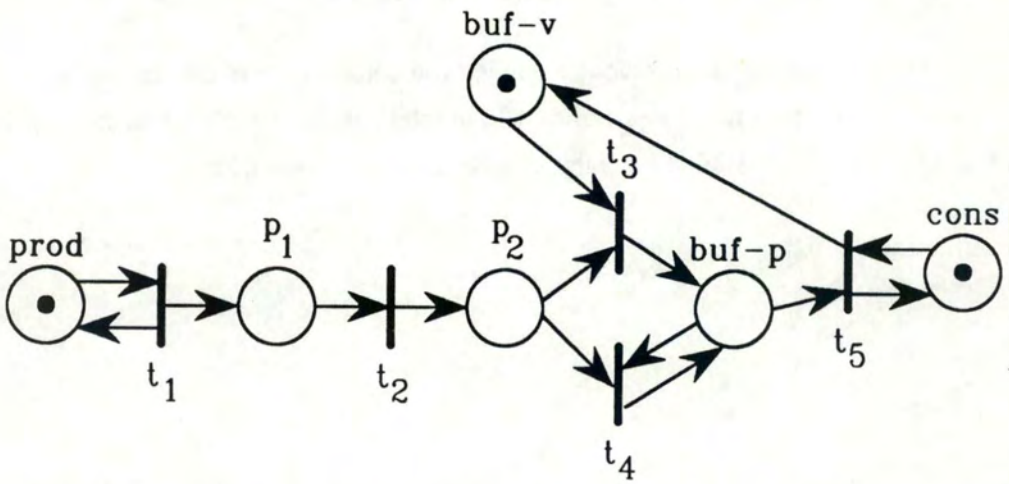


Figure 5.5. Réseau de Petri classique modélisant le protocole de transfert de données à sens unique

5.3. Applications

Dans cette section, nous analysons *les protocoles de transfert de données à sens unique* et *du bit alterné*.

Ces exemples n'ont d'autre but que de montrer de manière concrète l'utilité des réseaux de Petri temporels pour la modélisation de protocoles.

Remarquons que ces exemples étant de taille limitée, nous n'avons pas eu recours à la découpe en modules telle que présentée ci-dessus.

5.3.1. Le protocole de transfert de données à sens unique

A) Présentation

Le *protocole de transfert de données à sens unique* est un protocole assez simple. On y considère une entité productrice qui envoie des messages et une entité réceptrice qui reçoit ces mêmes messages [JUA 89].

Dans une première étape, nous allons modéliser ce protocole à l'aide des réseaux de Petri classiques, ce qui nous permettra de montrer les limites de ce formalisme.

Ces limites vont nous amener à recourir à la représentation du protocole sous forme de réseaux temporels.

La construction du graphe des classes d'états à l'aide du simulateur présenté au chapitre précédent, nous permettra alors d'analyser différents comportements du protocole, engendrés par une modification des temporisations.

B) Modélisation du protocole

1° Modèle des réseaux de Petri classiques

. Modélisation (Cf. figure 5.5.)

La place *prod* représente l'entité productrice prête à émettre un message. La transition t_1 modélise l'envoi du message et le retour de l'entité productrice dans l'état prêt à émettre, représenté par la place *prod*. La place p_1 représente le message envoyé, la transition t_2 la propagation du message sur le bus et la place p_2 un message arrivant sur le site de l'entité consommatrice.

Du côté du site du consommateur, la place *cons* représente l'entité consommatrice prête à consommer un message. Avant d'être consommé, le message est mis dans un buffer : celui-ci, à

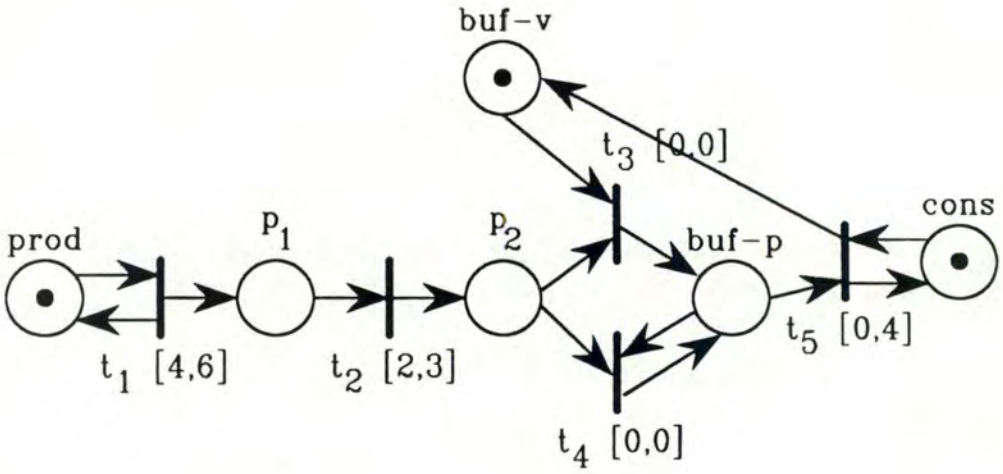


Figure 5.6. Réseau de Petri temporel modélisant le protocole de transfert de données à sens unique (modèle 1)

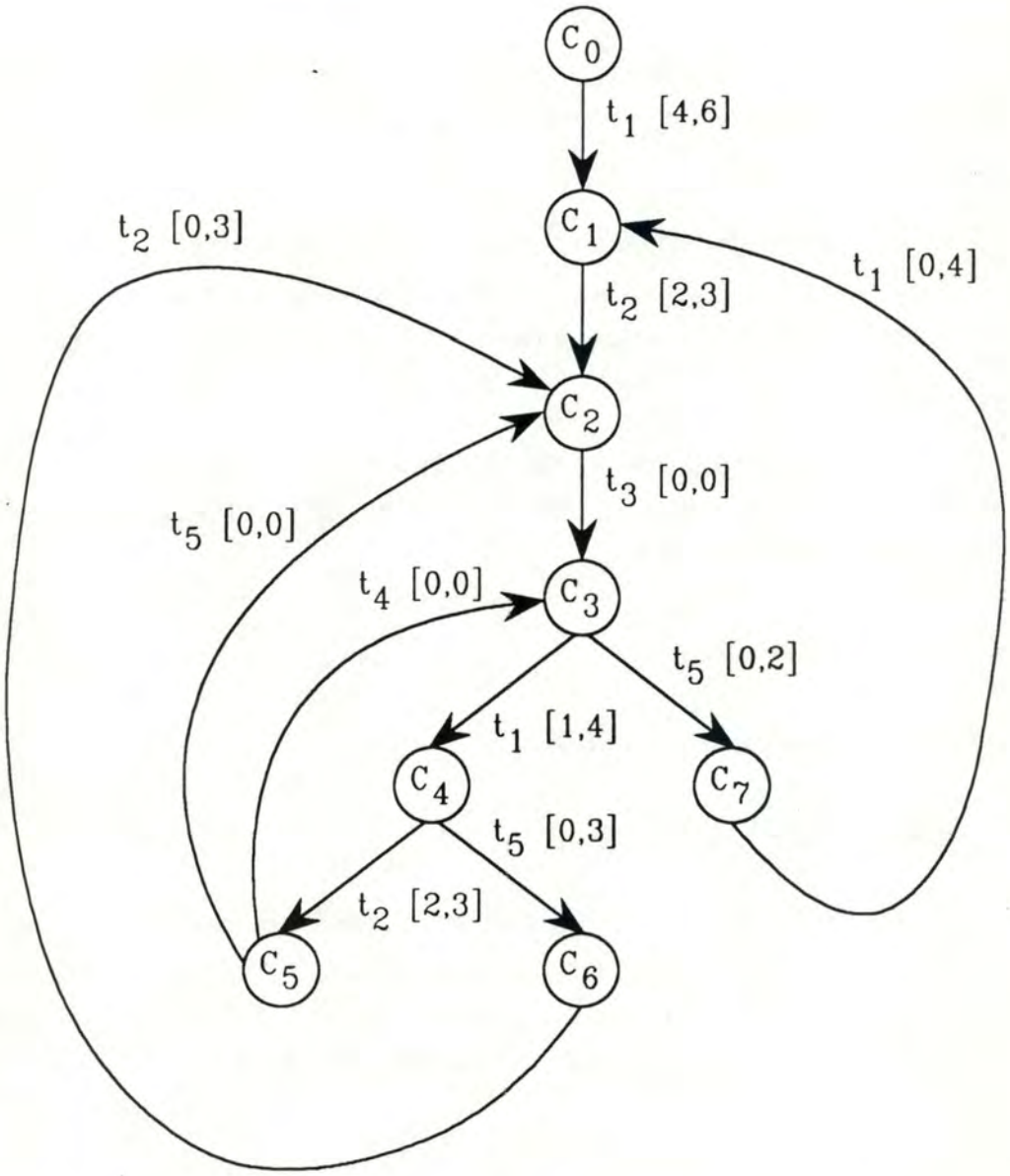


Figure 5.7. Graphe des classes d'états du modèle 1

l'arrivée du message, peut être vide (place *buf_v*) ou plein (place *buf_p*). La transition t_5 modélise la consommation du message et le retour du consommateur à l'état prêt à consommer (place *cons*).

La transition t_3 représente la situation normale d'arrivée d'un message et le stockage de celui-ci dans le buffer vide qui devient alors plein (passage du jeton de la place *buf_v* à la place *buf_p*).

Quant à la transition t_4, elle modélise une situation anormale sur le site du consommateur, à savoir l'arrivée d'un message alors que le buffer est encore plein. L'ancienne valeur est alors écrasée.

Comme il n'y a pas de contrôle de flux, il est important de modéliser cette situation anormale. Ainsi, nous pouvons dire que le système n'aura un comportement correct que si la transition t_4 n'est jamais tirée.

. Analyse

Le réseau représenté à la figure 5.5. est vivant, toutes les transitions peuvent en effet être tirées, mais non borné car la transition t_1 peut être tirée indéfiniment et de manière répétitive. Pour maîtriser le tir répétitif de t_1 tout en concervant la structure du réseau, nous ne voyons d'autres moyens que l'introduction de contraintes temporelles. Pour ce faire, nous utilisons le formalisme du modèle de Merlin.

2° Modèle des réseaux de Petri temporels

. Modélisation

La structure du réseau est identique à celle des réseaux de Petri classiques. Pour obtenir le réseau temporel (Cf. figure 5.6.), il suffit d'associer à chaque transition les intervalles de tir suivants :

- $t_1 = [4,6]$
- $t_2 = [2,3]$
- $t_3 = [0,0]$
- $t_4 = [0,0]$
- $t_5 = [0,4]$

. Analyse

L'approche énumérative permet d'obtenir le graphe des classes d'états représenté à la figure 5.7. Ce graphe est borné à 8 classes d'états et vivant (la transition t_4 peut être tirée).

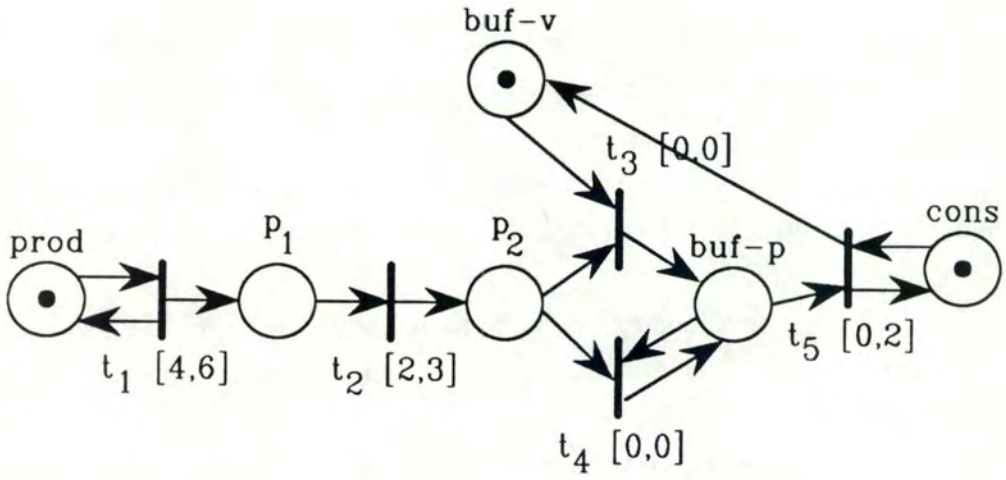


Figure 5.8. Réseau de Petri temporel modélisant le protocole de transfert de données à sens unique (modèle 2)

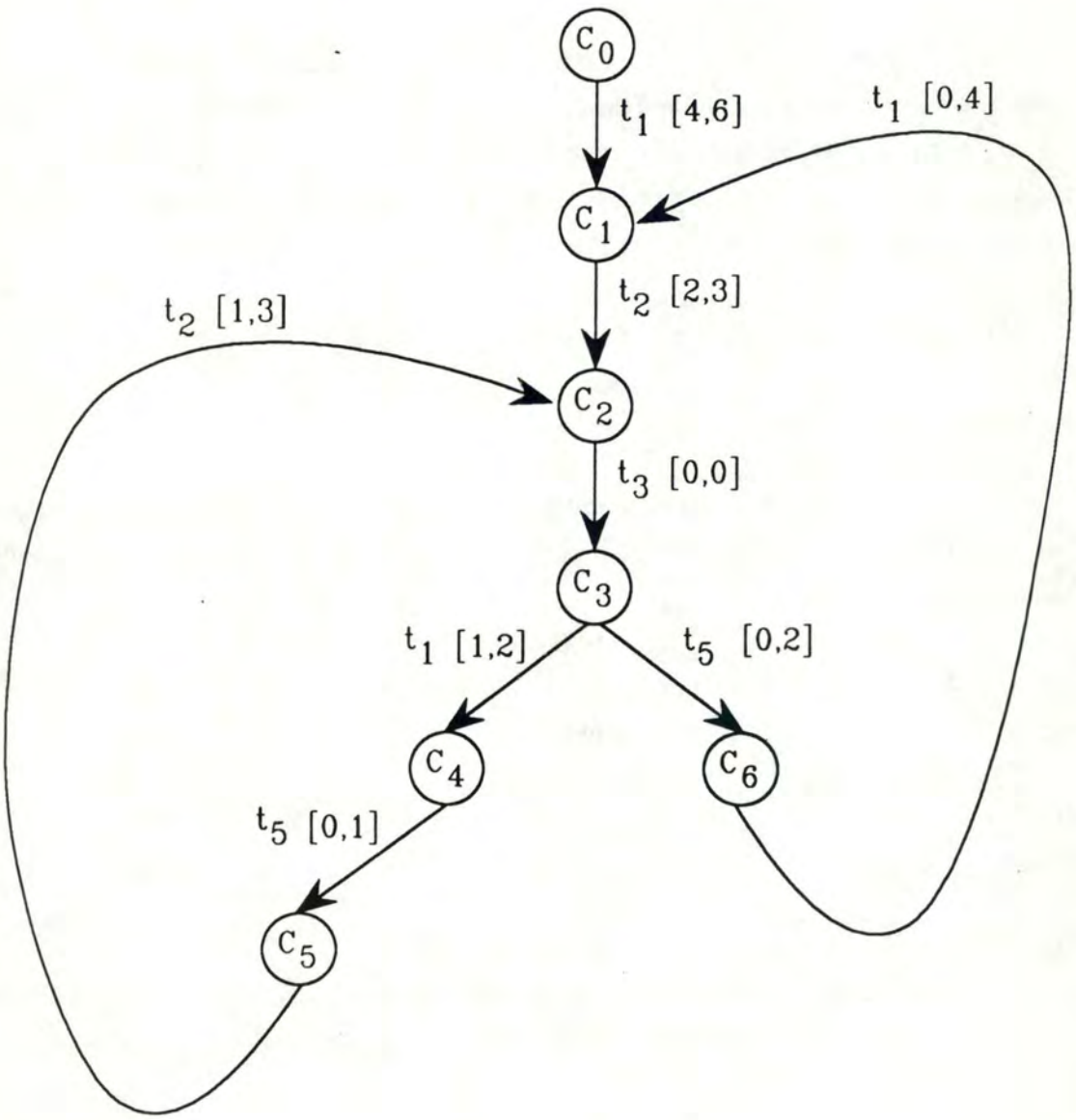


Figure 5.9. Graphe des classes d'états du modèle 2

Le contenu des classes d'états est donné par :

$C_0 = (M_0,D_0)$:
M_0 : prod(1), buf_v(1), cons(1)
D_0 : $4 \leq t_1 \leq 6$

$C_1 = (M_1,D_1)$:
M_1 : prod(1), p_1(1), buf_v(1), cons(1)
D_1 : $4 \leq t_1 \leq 6$
$2 \leq t_2 \leq 3$

$C_2 = (M_2,D_2)$:
M_2 : prod(1), p_2(1), buf_v(1), cons(1)
D_2 : $1 \leq t_1 \leq 4$
$0 \leq t_3 \leq 0$

$C_3 = (M_3,D_3)$:
M_3 : prod(1), buf_p(1), cons(1)
D_3 : $1 \leq t_1 \leq 4$
$0 \leq t_5 \leq 4$

$C_4 = (M_4,D_4)$:
M_4 : prod(1), p_1(1), buf_p(1), cons(1)
D_4 : $4 \leq t_1 \leq 6$
$2 \leq t_2 \leq 3$
$0 \leq t_5 \leq 3$

$C_5 = (M_5,D_5)$:
M_5 : prod(1), p_2(1), buf_p(1), cons(1)
D_5 : $1 \leq t_1 \leq 4$
$0 \leq t_4 \leq 0$
$0 \leq t_5 \leq 1$
$1 \leq t_1 - t_2 \leq 6$

$C_6 = (M_6,D_6)$:
M_6 : prod(1), p_1(1), buf_v(1), cons(1)
D_6 : $0 \leq t_2 \leq 3$
$1 \leq t_1 \leq 6$
$0 \leq t_1 - t_2 \leq 4$

$C_7 = (M_7,D_7)$:
M_7 : prod(1), buf_v(1), cons(1)
D_7 : $0 \leq t_1 \leq 4$

L'analyse du graphe des classes d'états nous amène à formuler les remarques suivantes :

- les contraintes temporelles permettent d'atteindre la propriété borné du réseau,
- la transition t_4 peut être tirée; ceci signifie que les temps de production et de consommation sont tels que quand un message arrive à l'entité consommatrice, le buffer est encore plein.

Corrigeons cette situation anormale en rendant l'entité consommatrice plus rapide, c'est-à-dire en donnant un nouvel intervalle de tir à la transition qui modélise l'action de consommation; soit $t_5 = [0,2]$ (Cf. figure 5.8.).

Le nouveau graphe des classes d'états obtenu est représenté à la figure 5.9. Il est borné à 7 classes d'états et non vivant : la transition t_4 ne peut jamais être tirée.

Le contenu des classes d'états est donné par :

$C_0 = (M_0,D_0)$:
M_0 : prod(1), buf_v(1), cons(1)
D_0 : $4 \leq t_1 \leq 6$

$C_1 = (M_1,D_1)$:
M_1 : prod(1), p_1(1), buf_v(1), cons(1)
D_1 : $4 \leq t_1 \leq 6$
$2 \leq t_2 \leq 3$

$C_2 = (M_2, D_2)$:
M_2 : prod(1), p_2(1), buf_v(1), cons(1)
D_2 : $1 \leq t_1 \leq 4$
$0 \leq t_3 \leq 0$

$C_3 = (M_3, D_3)$:
M_3 : prod(1), buf_p(1), cons(1)
D_3 : $1 \leq t_1 \leq 4$
$0 \leq t_5 \leq 2$

$C_4 = (M_4, D_4)$:
M_4 : prod(1), p_1(1), buf_p(1), cons(1)
D_4 : $4 \leq t_1 \leq 6$
$2 \leq t_2 \leq 3$
$0 \leq t_5 \leq 1$

$C_5 = (M_5, D_5)$:
M_5 : prod(1), p_1(1), buf_v(1), cons(1)
D_5 : $3 \leq t_1 \leq 6$
$1 \leq t_2 \leq 3$
$1 \leq t_1 - t_2 \leq 4$

$C_6 = (M_6, D_6)$:
M_6 : prod(1), buf_v(1), cons(1)
D_6 : $0 \leq t_1 \leq 4$

Quand un message arrive à l'entité consommatrice, le message précédent a déjà été consommé et le buffer est vide.

Une analyse plus approfondie du graphe des classes d'états montre l'existence de deux boucles ($C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_6 \rightarrow C_1$) et ($C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow C_5 \rightarrow C_2$). Ces deux boucles montrent le manque de synchronisation entre les entités productrice et consommatrice.

La première boucle montre qu'après que le message ait été mis dans le buffer (transition t_3), il est consommé avant qu'un nouveau ne soit produit.

L'autre boucle, montre qu'un nouveau message est produit (transition t_1) avant la consommation du message en cours. Toutefois, ce dernier est consommé avant l'arrivée du nouveau message dans le buffer.

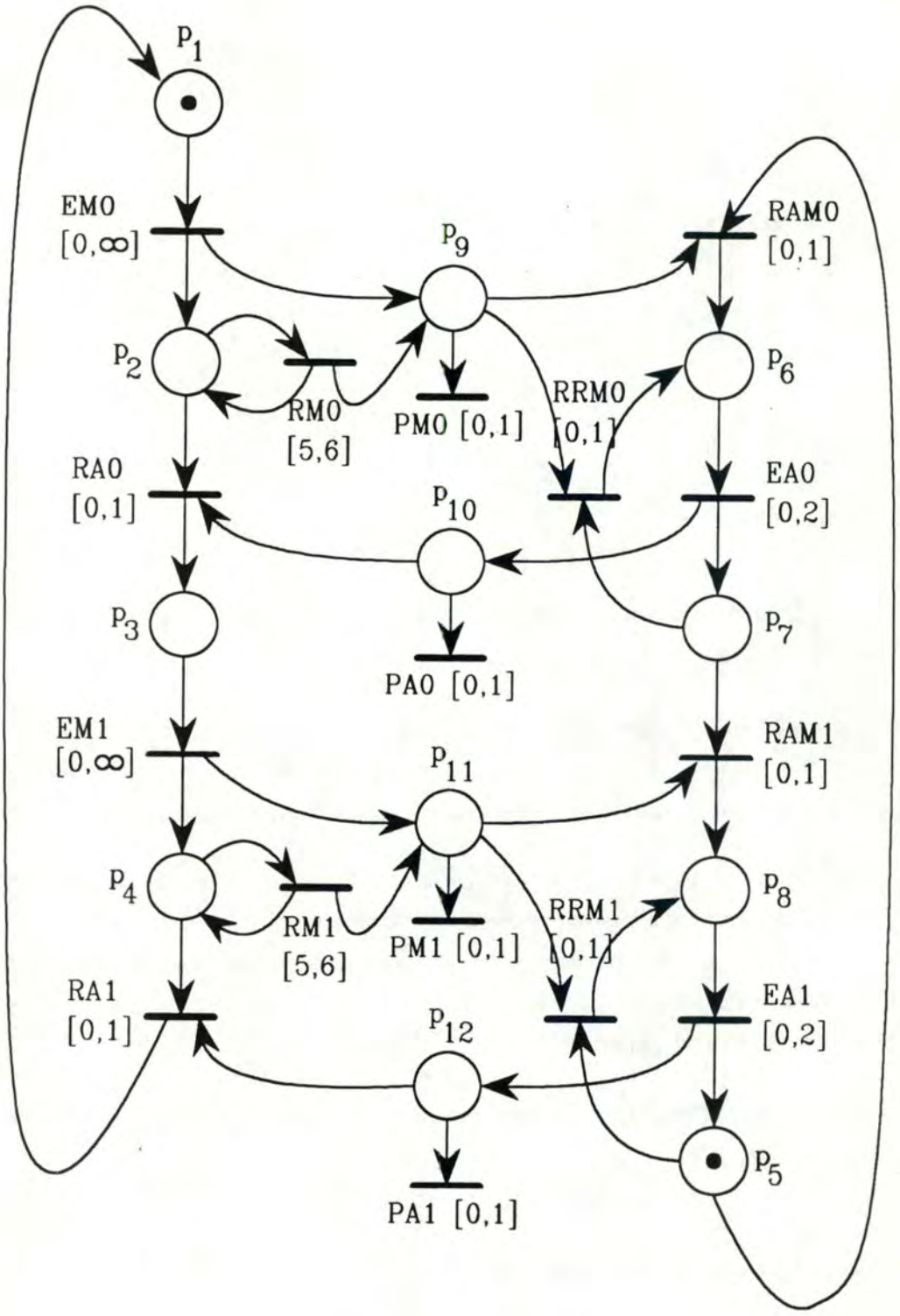
Le graphe des classes d'états montre que le système évolue correctement et le protocole est ainsi vérifié.

5.3.2. Le protocole du bit alterné

A) Présentation

Le *protocole du bit alterné* est sans aucun doute l'exemple le plus souvent cité dans la littérature des réseaux de Petri classiques.

Il nous a apparu également bien adapté pour montrer l'utilité des réseaux de Petri temporels pour la modélisation et la vérification des mécanismes permettant de corriger les pertes de messages.



EM0 : Emission Message 0
RAM0 : Réception et Acceptation Message 0
PM0 : Perte Message 0
RM0 : Réémission Message 0
RRM0 : Réception et Rejet Message 0
EA0 : Emission Accusé 0
RA0 : Réception Accusé 0
PA0 : Perte Accusé 0

EM1 : Emission Message 1
RAM1 : Réception et Acceptation Message 1
PM1 : Perte Message 1
RM1 : Réémission Message 1
RRM1 : Réception et Rejet Message 1
EA1 : Emission Accusé 1
RA1 : Réception Accusé 1
PA1 : Perte Accusé 1

Figure 5.10. Le protocole du Bit Alterné

Ces mécanismes sont habituellement implantés à l'aide de temps-limites et, les réseaux de Petri temporels constituent un outil efficace pour montrer que ces temps-limites sont correctement établis.

Le protocole du bit alterné est un protocole de transfert de données du type envoi-attente. Un émetteur désire transmettre des messages à un récepteur. Il envoie le premier message après l'avoir étiqueté avec un bit égal à 0. Quand le récepteur reçoit le message, il renvoie à l'émetteur un accusé de réception avec un bit identique à celui du message reçu. Si au bout d'un certain temps (expiration de la temporisation), l'émetteur n'a pas reçu d'accusé de réception, il renvoie son message car celui-ci a pu se perdre; ... et cela, jusqu'à ce qu'il ait reçu l'accusé. A ce moment là, l'émetteur sait qu'il peut envoyer le message suivant avec un bit égal à 1 (alternance par rapport au bit précédent). L'envoi de messages se poursuit ainsi en alternant les bits qui y sont associés.

Le mécanisme de retransmission permet de corriger les pertes de messages mais peut introduire des duplications à tort de messages, lorsque le message a été reçu par le récepteur mais que l'accusé de réception s'est perdu. Dès lors, la numérotation des messages (bits 0 et 1) est ici très importante car elle permet au récepteur de voir si le message qu'il reçoit est un nouveau message ou une copie du dernier message qu'il a accepté.

B) Modélisation du protocole

La figure 5.10. montre un réseau de Petri modélisant le protocole du bit alterné.

Il est possible d'y reconnaître trois processus : l'émetteur, le milieu de transmission et le récepteur.

L'émetteur peut être dans quatre états différents modélisés par les places :

- p_1 : état de repos avant l'envoi du message 0.
- p_2 : état d'attente de l'accusé de réception relatif au message 0.
- p_3 : état de repos avant l'envoi du message 1.
- p_4 : état d'attente de l'accusé de réception relatif au message 1.

Il peut exécuter six opérations qui sont modélisées par les transitions suivantes :

- **EM0** : envoi du message 0, par l'intermédiaire du milieu de transmission, et armement du temps-limite.
- **RM0** : déclenchement du temps-limite, c'est-à-dire expiration de la temporisation, réémission du message 0 et réarmement du temps-limite.
- **RA0** : réception de l'accusé de réception relatif au message 0.
- **EM1** : envoi du message 1, par l'intermédiaire du milieu de transmission, et armement du temps-limite.

- **RM1** : déclenchement du temps-limite, réémission du message 1 et réarmement du temps-limite.
- **RA1** : réception de l'accusé de réception relatif au message 1.

Le récepteur peut être dans quatre états différents :

- p_5 : état d'attente du message 0.
- p_6 : état prêt à émettre l'accusé de réception relatif au message 0.
- p_7 : état d'attente du message 1.
- p_8 : état prêt à émettre l'accusé de réception relatif au message 1.

Il peut exécuter les six actions suivantes :

- **RAM0** : réception et acceptation du message 0.
- **EA0** : émission de l'accusé de réception relatif au message 0.
- **RRM0** : réception et rejet du message 0.
- **RAM1** : réception et acceptation du message 1.
- **EA1** : émission de l'accusé de réception relatif au message 1
- **RRM1** : réception et rejet du message 1

Quant au milieu de transmission, il peut être dans cinq états dont quatre sont représentés par les places qui suivent :

- p_9 : transit du message 0.
- p_{10} : transit de l'accusé de réception relatif au message 0.
- p_{11} : transit du message 1.
- p_{12} : transit de l'accusé de réception relatif au message 1.

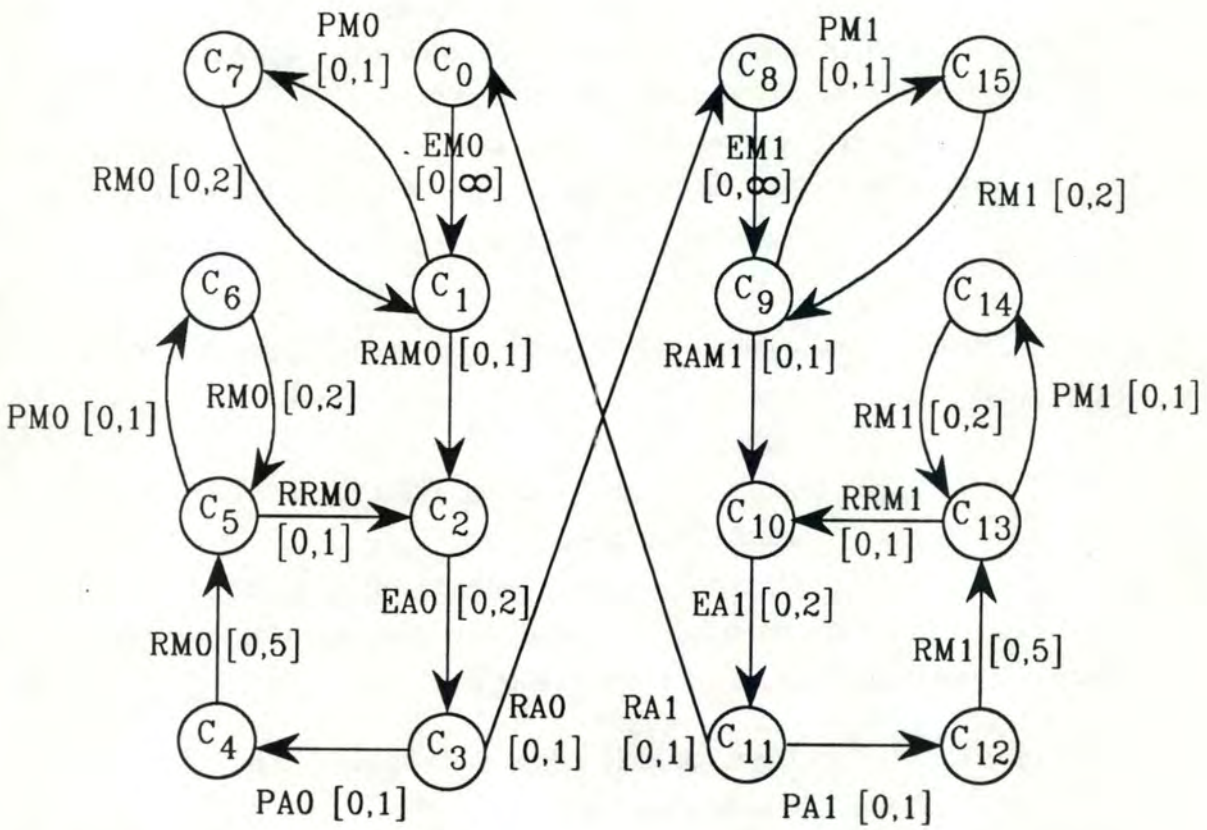
Le cinquième état correspond à un **milieu de transmission vide**, ce qui est modélisé par l'absence de jeton dans les quatre places citées ci-dessus.

Les seules actions que le milieu de transmission peut exécuter sont :

- PM0 : perte du message 0.
- PA0 : perte de l'accusé de réception relatif au message 0.
- PM1 : perte du message 1.
- PA1 : perte de l'accusé de réception relatif au message 1.

Sur le réseau, les pertes de messages et d'accusés de réception sont simplement représentées par des transitions qui n'ont pas de places de sortie.

Dans un but de simplification, les messages endommagés sont assimilés à des messages perdus.



EM0 : Emission Message 0
RAM0 : Réception et Acceptation Message 0
PM0 : Perte Message 0
RM0 : Réémission Message 0
RRM0 : Réception et Rejet Message 0
EA0 : Emission Accusé 0
RA0 : Réception Accusé 0
PA0 : Perte Accusé 0

EM1 : Emission Message 1
RAM1 : Réception et Acceptation Message 1
PM1 : Perte Message 1
RM1 : Réémission Message 1
RRM1 : Réception et Rejet Message 1
EA1 : Emission Accusé 1
RA1 : Réception Accusé 1
PA1 : Perte Accusé 1

Figure 5.11. Graphe des classes d'états pour le protocole du Bit Alterné

Toute perte de message ou d'accusé de réception peut se produire dès que le message ou l'accusé est déposé sur le milieu de transmission et ce, jusqu'à la durée de vie maximale du message ou de l'accusé de réception dans le milieu de transmission. Cette durée est notamment fonction du temps de propagation entre l'émetteur et le récepteur.

Pour que le protocole ait un fonctionnement correct, il faut que la durée de vie maximale des messages dans le milieu de transmission soit connue d'avance, et que le temps-limite attendu avant de réémettre un message perdu soit fixé en fonction de cette durée de vie; ceci parce qu'il ne faut pas que le récepteur interprète des vieux messages comme des nouveaux.

En général, on choisira le temps-limite tel qu'il soit supérieur à la durée de vie maximale des messages dans le milieu de transmission, plus le temps nécessaire au récepteur pour envoyer un accusé de réception, plus la durée de vie maximale de ce dernier.

Pour modéliser le protocole, nous supposons que la durée de vie d'un message ou d'un accusé de réception sur le milieu de transmmission est comprise dans l'intervalle [0,1] et que le temps nécessaire au récepteur pour envoyer un accusé de réception est compris dans l'intervalle [0,2].

En tenant compte de ce qui a été dit, nous considérons que la retransmission d'un message peut se produire à n'importe quel instant compris dans l'intervalle [5,6] après avoir envoyé la dernière copie du message.

Quant à l'émission de la première copie de chaque message, nous supposons qu'il n'y a pas de contraintes de dates et nous attribuons aux transitions correspondantes l'intervalle $[0,\infty]$.

C) Analyse

Le graphe des classes d'états produit à partir du réseau de Petri est représenté à la figure 5.11. Comme on peut le constater, le réseau est borné à 16 classes d'états.

$C_0 = (M_0,D_0)$:
$M_0 : p_1(1), p_5(1)$
$D_0 : 0 \leq \underline{EM0} \leq \infty$

$C_1 = (M_1,D_1)$:
$M_1 : p_2(1), p_5(1), p_9(1)$
$D_1 : 5 \leq \underline{RM0} \leq 6$
$0 \leq \underline{RAM0} \leq 1$
$0 \leq \underline{PM0} \leq 1$

$C_2 = (M_2,D_2)$:
$M_2 : p_2(1), p_6(1)$
$D_2 : 4 \leq \underline{RM0} \leq 6$
$0 \leq \underline{EA0} \leq 2$

$C_3 = (M_3,D_3)$:
$M_3 : p_2(1), p_7(1), p_{10}(1)$
$D_3 : 2 \leq \underline{RM0} \leq 6$
$0 \leq \underline{RA0} \leq 1$
$0 \leq \underline{PA0} \leq 1$

$C_4 = (M_4,D_4)$:
$M_4 : p_2(1), p_7(1)$
$D_4 : 0 \leq \underline{RM0} \leq 5$

$C_5 = (M_5,D_5)$:
$M_5 : p_2(1), p_7(1), p_9(1)$
$D_5 : 5 \leq \underline{RM0} \leq 5$
$0 \leq \underline{RRM0} \leq 1$
$0 \leq \underline{PM0} \leq 1$

$C_6 = (M_6, D_6)$:
$M_6 : p_2(1), p_7(1)$
$D_6 : 0 \leq \underline{RM0} \leq 2$

$C_7 = (M_7, D_7)$:
$M_7 : p_2(1), p_5(1)$
$D_7 : 0 \leq \underline{RM0} \leq 2$

$C_8 = (M_8, D_8)$:
$M_8 : p_3(1), p_7(1)$
$D_8 : 0 \leq \underline{EM1} \leq \infty$

$C_9 = (M_9, D_9)$:
$M_9 : p_4(1), p_7(1), p_{11}(1)$
$D_9 : 5 \leq \underline{RM1} \leq 6$
$0 \leq \underline{RAM1} \leq 1$
$0 \leq \underline{PM1} \leq 1$

$C_{10} = (M_{10}, D_{10})$:
$M_{10} : p_4(1), p_8(1)$
$D_{10} : 4 \leq \underline{RM1} \leq 6$
$0 \leq \underline{EA1} \leq 2$

$C_{11} = (M_{11}, D_{11})$:
$M_{11} : p_4(1), p_5(1), p_{12}(1)$
$D_{11} : 2 \leq \underline{RM1} \leq 6$
$0 \leq \underline{RA1} \leq 1$
$0 \leq \underline{PA1} \leq 1$

$C_{12} = (M_{12}, D_{12})$:
$M_{12} : p_4(1), p_5(1)$
$D_{12} : 0 \leq \underline{RM1} \leq 5$

$C_{13} = (M_{13}, D_{13})$:
$M_{13} : p_4(1), p_5(1), p_{11}(1)$
$D_{13} : 5 \leq \underline{RM1} \leq 6$
$0 \leq \underline{RRM1} \leq 1$
$0 \leq \underline{PM1} \leq 1$

$C_{14} = (M_{14}, D_{14})$:
$M_{14} : p_4(1), p_5(1)$
$D_{14} : 0 \leq \underline{RM1} \leq 2$

$C_{15} = (M_{15}, D_{15})$:
$M_{15} : p_4(1), p_7(1)$
$D_{15} : 0 \leq \underline{RM1} \leq 2$

Le graphe des classes d'états montre clairement que le récepteur ne peut accepter plus d'une copie de chaque message (les transitions RAM0 et RAM1 alternent dans tous les chemins du graphe) et le transfert de messages peut avoir lieu (le réseau est vivant).

La boucle $C_1 \rightarrow C_7 \rightarrow C_1$ ($C_9 \rightarrow C_{15} \rightarrow C_9$) représente la perte du message 0 (respectivement la perte du message 1) et la réémission de celui-ci.

La boucle $C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow C_5 \rightarrow C_2$ ($C_{10} \rightarrow C_{11} \rightarrow C_{12} \rightarrow C_{13} \rightarrow C_{10}$) représente la perte de l'accusé de réception 0 (respectivement la perte de l'accusé de réception 1) et la réémission du message. La boucle $C_5 \rightarrow C_6 \rightarrow C_5$ (respectivement la boucle $C_{13} \rightarrow C_{14} \rightarrow C_{13}$) représente la perte possible du message lorsque celui-ci est réémis.

Quand le message est réémis suite à la perte de l'accusé de réception, on ne passe plus par la transition RAM0 (ou par la transition RAM1) mais par la transition RRM0 (respectivement par la transition RRM1) car le message est une recopie du dernier message que le récepteur a déjà accepté. Dès lors, lorsque le récepteur reçoit ce message, non seulement il le rejette, mais il envoie également un nouvel accusé de réception. Ainsi, la transition RRM0 (respectivement RRM1) permet non seulement d'envoyer un nouvel accusé de réception, mais aussi d'éviter la duplication à tort du message.

L'examen du contenu des classes d'états montre qu'un message au plus est en transit à chaque instant (les places p_9, p_{10}, p_{11}, p_{12} contiennent au plus un jeton pour tout marquage accessible).

Dès lors, le fonctionnement obtenu correspond bien aux spécifications, les messages acceptés par le récepteur sont bien les messages émis par l'émetteur et sont acceptés, sans duplication, dans l'ordre de leur émission.

5.3.3. Quelques remarques

Le premier exemple que nous avons développé a montré la nécessité d'intégrer le temps dans l'analyse des systèmes distribués.

L'introduction de temporisations nous a permis d'obtenir un réseau borné et donc analysable, ce qui n'était pas le cas avec le modèle des réseaux de Petri classiques.

Ceci nous permet d'affirmer que contrairement au modèle des réseaux de Petri classiques, le modèle des réseaux de Petri temporels est adéquat pour modéliser le comportement de systèmes dans lesquels les mécanismes de synchronisation ne se font pas par acquittement de messages mais par des contraintes temporelles comme dans le cas, par exemple, du protocole de transfert de données à sens unique.

Quant au protocole du bit alterné, une modélisation par les réseaux de Petri classiques est proposée dans [AYA 85b].

Dans cette modélisation, des places et transitions auxiliaires sont utilisées pour exprimer les effets des contraintes temporelles; car, si l'utilisation des réseaux de Petri classiques ne permet pas d'introduire explicitement le paramètre temps, les causes qui provoquent l'expiration des temporisations peuvent être simulées.

Toutefois, la comparaison des deux modèles montre clairement que les réseaux temporels de Merlin permettent une spécification beaucoup plus naturelle, les contraintes temporelles étant simplement exprimées par les intervalles de tir associés aux transitions et sans recours à des mécanismes artificiels.

Ceci a pour effet de réduire la complexité de la représentation et du graphe d'accessibilité. L'analyse devient dès lors beaucoup plus commode à réaliser.

5.4. Conclusion

Dans ce chapitre, nous avons présenté une méthodologie d'analyse permettant d'exploiter au mieux le modèle des réseaux de Petri temporels pour l'analyse des protocoles de communication.

Nous avons montré par des exemples le domaine d'application ainsi que l'utilité du modèle et de la méthode par énumération des classes d'états.

Ceci nous a également permis de dégager les avantages des réseaux de Petri temporels par rapport aux réseaux de Petri classiques.

Le chapitre suivant concerne une application significative du modèle à la modélisation et à la vérification du réseau local industriel FIP.

Cette application de plus grande complexité que les exemples traités jusqu'ici, outre les résultats apportés par l'analyse, sera l'occasion de mettre à jour d'autres caractéristiques des réseaux temporels et de mettre en évidence certaines de leurs limites.

CHAPITRE 6

SPECIFICATION FORMELLE DU PROTOCOLE DE SCRUTATION CYCLIQUE DU RESEAU FIP

6.1. Introduction

Ce chapitre présente une approche pour modéliser et analyser le fonctionnement du protocole de scrutation cyclique du réseau industriel FIP.

Le modèle utilisé est celui des réseaux de Petri temporels, auquel nous appliquons la méthode d'analyse par énumération des classes d'états.

Nous décrivons dans la section qui suit le protocole FIP, qui est un protocole de communication conçu pour des échanges d'informations entre capteurs, actionneurs et automates programmables à l'intérieur de processus industriels.

Ensuite, nous analysons les contraintes temporelles pour le protocole de liaison de données de FIP, relatif à la scrutation cyclique de variable (service périodique).

Les objectifs principaux de ce chapitre sont, d'une part, de montrer par un exemple concret que les réseaux de Petri temporels permettent d'exprimer commodément les contraintes temporelles des protocoles de communication et, d'autre part, de tirer les conclusions de l'analyse de la couche liaison de FIP.

En annexe, le lecteur trouvera l'architecture du modèle que nous avons implémenté sous RdP et qui nous a permis de mener à bien notre analyse.

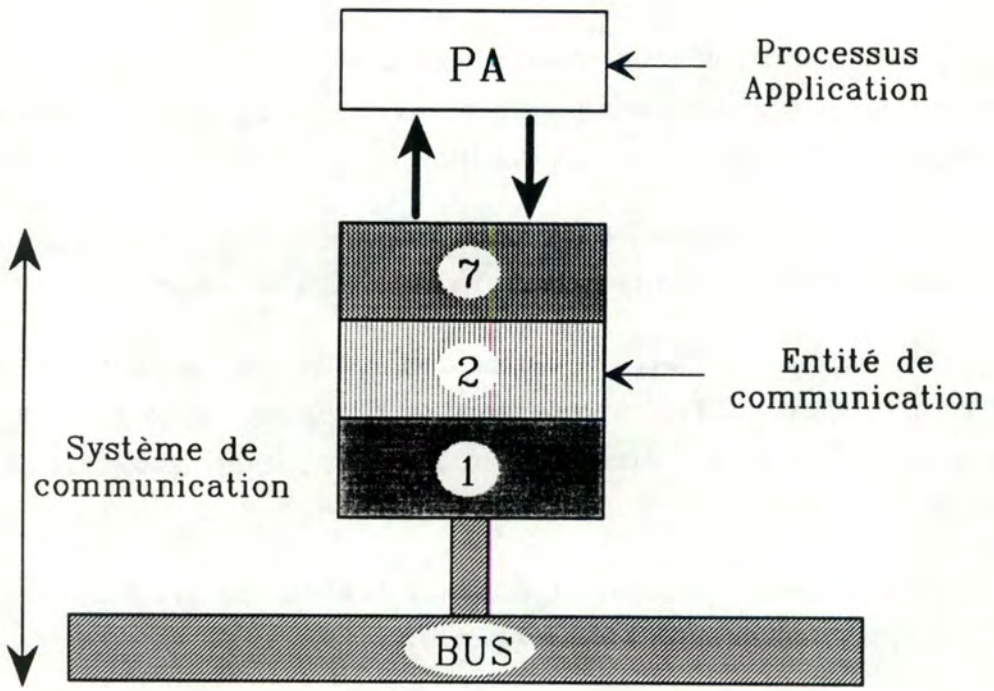


Figure 6.1. Système de communication de FIP

6.2. Présentation de FIP

[AZE 88d] [BAR 89] [CIN 89] [FIP 88a] [FIP 88b] [FIP 89] [HAS 89] [LET 89]

Le besoin d'interconnecter les différents éléments d'une installation industrielle automatisée a conduit à définir un ensemble de conventions, appelées protocoles, pour régler, au travers de leur interface, les relations entre les comportements des différentes entités de l'activité.

A la fin 1982, un groupe de travail a été constitué à la demande de la Mission Scientifique et Technique du Ministère de la Recherche et de la Technologie. La mission de ce groupe fut d'analyser l'opportunité de promouvoir des réseaux locaux dans le domaine du contrôle et de la commande de processus.

En juin 1986 fut créée l'association "Club FIP", constituée des représentants de différentes sociétés d'utilisateurs, de fabricants et des laboratoires de recherche. Actuellement, le club possède un certain nombre de comités et sous-comités techniques pour traiter des problèmes divers relatifs aux couches de protocoles, au contrôle d'accès au support liaison de données, au lien physique et topologique, à la rédaction d'une proposition de norme, ...

Ce groupe de travail a fait le constat du besoin de projets et de normes en ce qui concerne les réseaux *bus de terrain* et a noté que dans la majorité des cas, ce type de communication était réalisé en liaisons point à point, ce qui représente d'importants coûts de câblage.

L'objectif de FIP fut donc de proposer un moyen de communication qui permette de réduire globalement le coût de câblage et d'améliorer ainsi la compétitivité du système tout en garantissant une meilleure qualité de la communication.

6.2.1. Le protocole FIP

FIP est un protocole pour des réseaux locaux industriels de type *bus de terrain*, conçu pour assurer les liaisons entre le niveau des capteurs et actionneurs et celui des automates programmables, régulateurs et autres équipements des automatismes. Il assure aussi bien la collecte de toutes les informations fournies par les capteurs aux automates, que la diffusion de tous les ordres fournis par les automates aux actionneurs répartis sur le terrain.

Ce protocole s'inspire du modèle de référence d'interconnexion des systèmes ouverts de l'ISO. De ce modèle, il reprend les couches physique (couche 1), liaison de données (couche 2) et application (couche 7) (Cf. figure 6.1.).

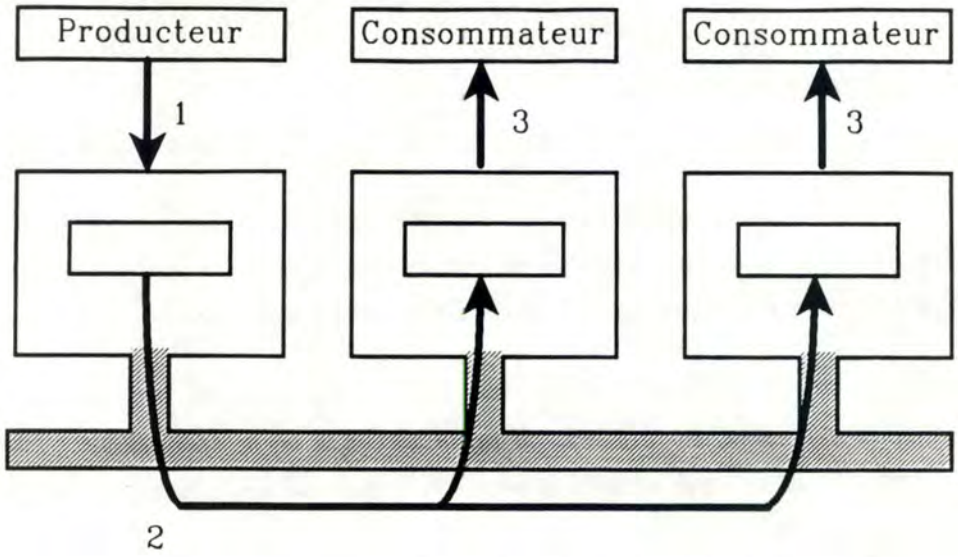


Figure 6.2. Opérations d'échange

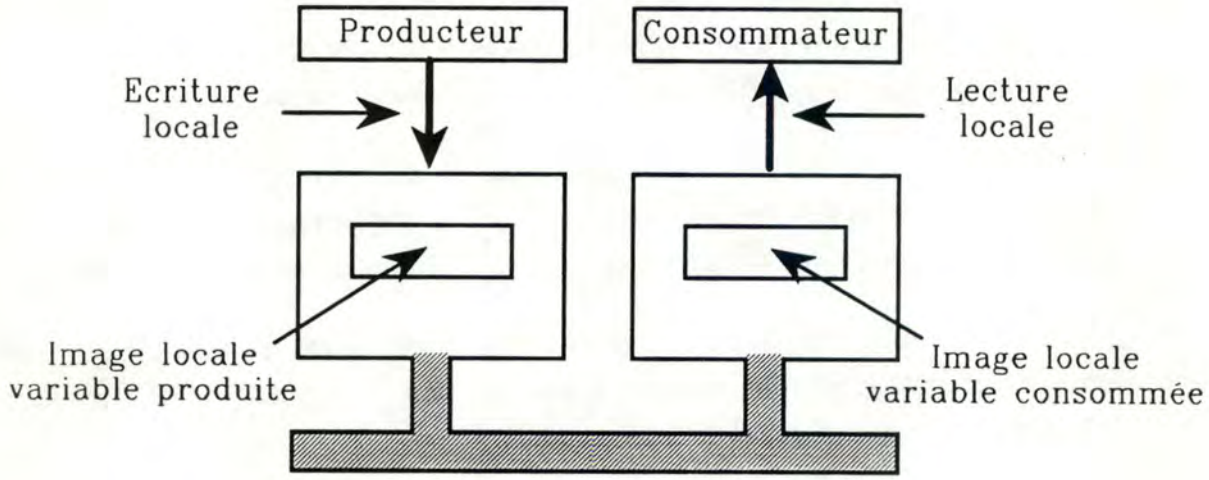


Figure 6.3. Lectures et écritures locales

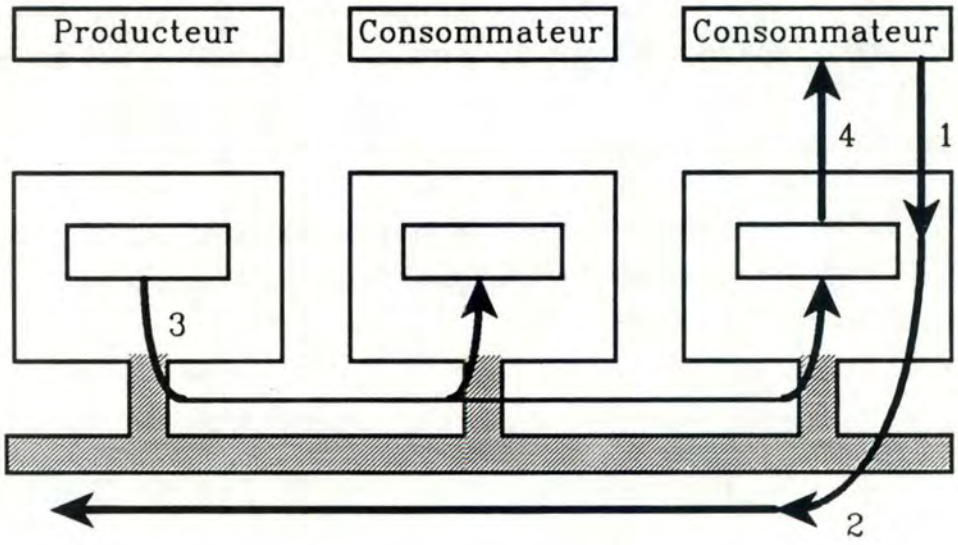


Figure 6.4. Mécanisme de demande de la valeur d'une variable

A) Présentation des trois couches

1° La couche application

La couche application offre au processus utilisateur un ensemble de services lui permettant des actions en lecture ou en écriture sur une variable. Ce mécanisme met en relation un producteur de l'information et un ou plusieurs utilisateurs.

Ainsi que le montre le schéma de la figure 6.2., trois opérations d'échange sont possibles :

1. Production d'une valeur de variable par un PA (processus d'application) producteur.
2. Transfert de cette valeur du producteur vers les consommateurs.
3. Consommation de la valeur par plusieurs PA consommateurs.

Grâce aux services d'écriture et de lecture locale, respectivement *l_put_demande* et *l_get_demande*, chaque processus d'application d'une application répartie, peut manipuler l'image qu'il possède d'une variable.

Ces requêtes d'écriture et de lecture locales sont respectivement déclenchées par les producteurs et les consommateurs de la variable (Cf. figure 6.3.).

Ces mécanismes, dits locaux, sont des mécanismes indépendants de l'activité du réseau qui n'entraînent aucun échange d'informations entre les entités raccordées au bus.

Par contre, lorsqu'un utilisateur veut connaître la valeur d'une variable présente dans une entité productrice, il effectue une demande de lecture distante. Cette valeur sera égale à la dernière valeur produite.

Les opérations seront décomposées de la manière suivante (Cf. figure 6.4.) :

1. Demande de lecture distante.
2. Demande de mise à jour de la valeur.
3. Transfert de la valeur de la variable du producteur vers le ou les consommateurs.
4. Confirmation de la lecture distante.

Les processus consommateurs sont informés de la réception d'une valeur de variable consommée (*l_read_ind*) tandis que les processus producteurs sont informés de l'émission de la valeur d'une variable produite (*l_send_ind*).

En plus de ces services, des informations booléennes, appelées promptitude et rafraîchissement, informent l'utilisateur sur la validité des données produites et consommées.

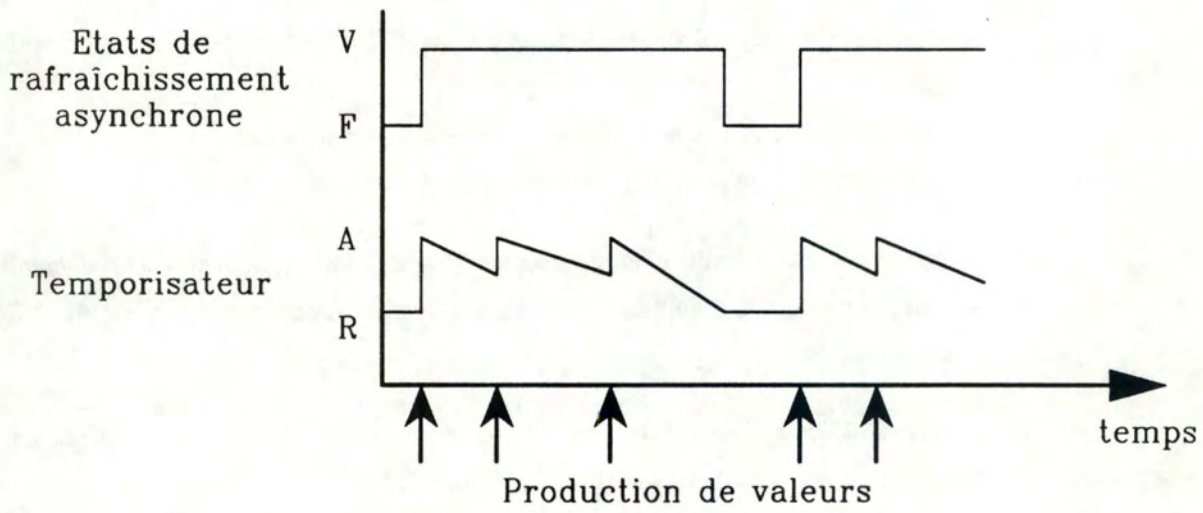


Figure 6.5. Rafraîchissement asynchrone

Les états de promptitude et de rafraîchissement peuvent être asynchrones, synchrones ou ponctuels.

a) La promptitude

La promptitude est une information de validité relative à la mise à disposition du consommateur, par le réseau, de la valeur d'une variable. Elle indique au consommateur si la variable consommée a été rafraîchie par le réseau depuis un temps inférieur à la période de consommation (période minimale à laquelle le consommateur accédera à la variable).

b) Le rafraîchissement

Le *rafraîchissement* est une information de validité relative à la mise à disposition du réseau par le producteur de la valeur d'une variable. Elle indique à un consommateur que le producteur de l'information a respecté ou n'a pas respecté un délai appelé période de production (période maximale à laquelle l'utilisateur producteur mettra la variable à disposition du réseau).

Considérons la valeur de l'état de rafraîchissement asynchrone par rapport aux instants de production de valeurs (Cf. figure 6.5.).

A la production d'une valeur, un temporisateur est armé (A) avec une valeur égale à la période de production. On constate que l'état de rafraîchissement asynchrone est vrai (V) tant que le temporisateur reste armé et devient faux (F) à l'expiration de celui-ci (R).

Les périodes de consommation et de production sont initialisées soit par l'écriture ou la réception de la valeur d'une variable (cas asynchrone) soit par la réception d'une variable de synchronisation (cas synchrone).

c) Service de synchronisation

Parmi les processus qui participent à une application répartie, il faut distinguer d'une part les *processus asynchrones* et d'autre part les *processus synchrones*.

Un *processus* est dit *asynchrone* si son exécution est indépendante du réseau. Par contre, un *processus synchrone* est un processus dont l'exécution est liée à une indication de réception en provenance du réseau.

Certains processus d'application ne supportent que le mode asynchrone. C'est pourquoi, il existe un *mécanisme de resynchronisation* qui permet à de tels processus de participer à une application répartie synchrone.

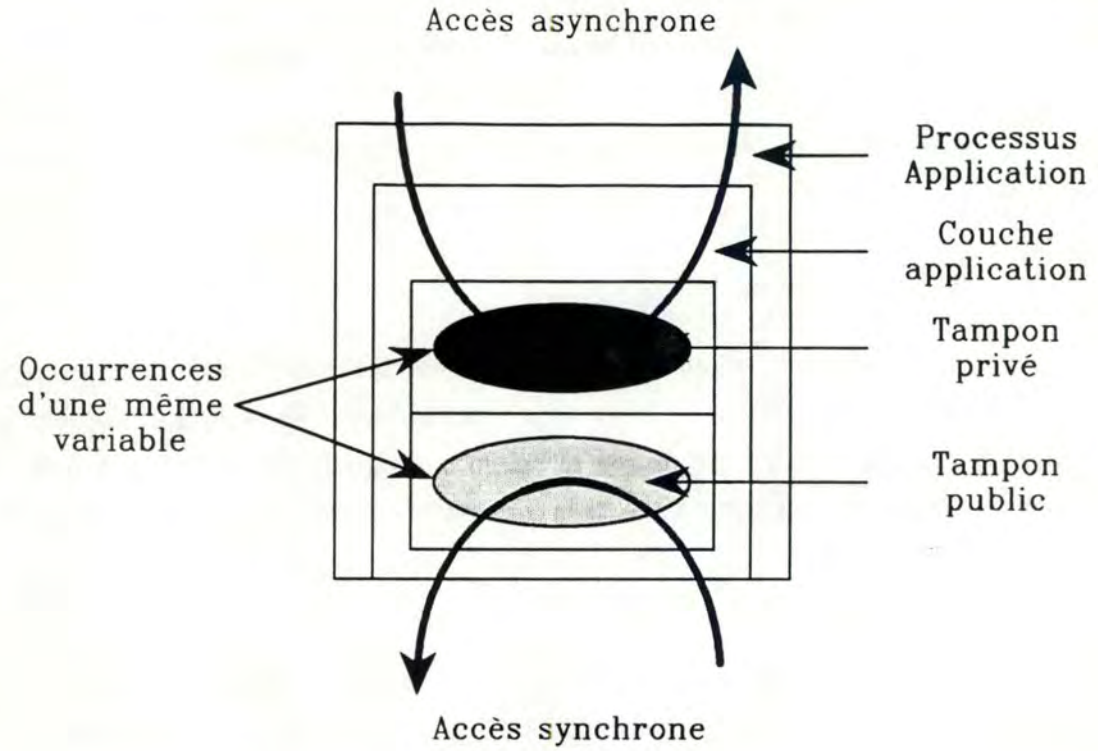
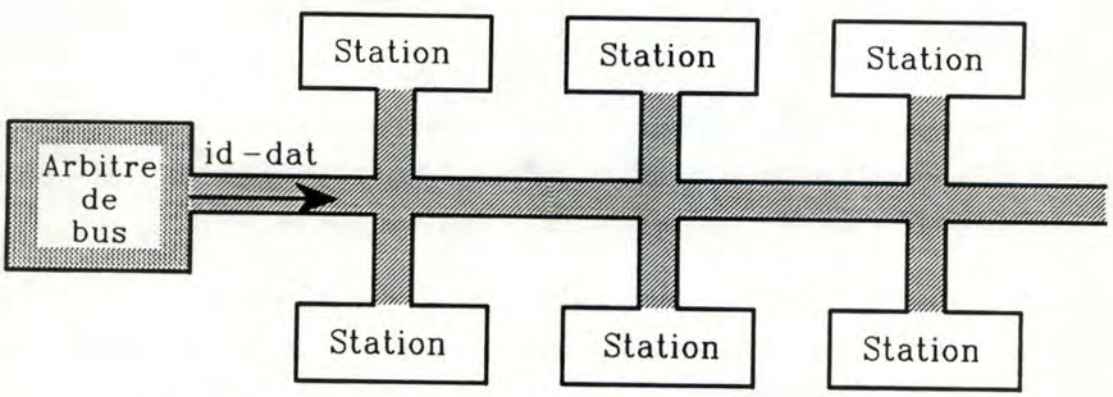
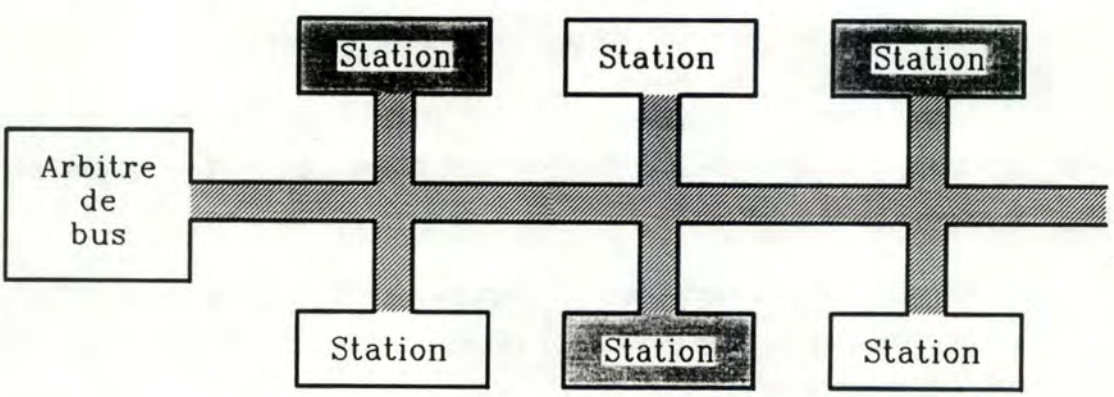


Figure 6.6. Services de resynchronisation



Phase 1 : l'arbitre de bus envoie un id-dat

Figure 6.7.



Phase 2 : certaines stations se reconnaissent comme productrices ou consommatrices de la valeur associée à l'id-dat

Figure 6.8.

Ce mécanisme (Cf. figure 6.6.) associe un deuxième tampon à une variable au niveau d'une entité application. La Mémoire Virtuelle Application constitue un tampon privé qui est accessible uniquement par le processus d'application. Il y accède grâce aux services d'écriture et de lecture locales asynchrones. La Mémoire Virtuelle Réseau qui constitue le tampon public n'est accessible que par le réseau.

2° *La couche liaison de données*

Les services que propose la norme FIP sont assez différents de ceux présents sur le marché. Le modèle général des communications s'appuie sur un modèle bipoint, c'est-à-dire que seules deux stations sont concernées : une station émettrice et une autre réceptive. Mais, ce mécanisme ne suffit pas dans le cadre des bus de terrain car une information produite peut être consommée par plusieurs utilisateurs. C'est pourquoi FIP propose un modèle s'appuyant sur la diffusion générale.

. Fonctionnement

FIP exploitant le principe de la diffusion générale, les informations traitées n'ont pas besoin d'être adressées à un destinataire. Le principe d'adressage utilisé est l'*adressage source* mais l'originalité de FIP est que l'adresse source est un identificateur représentant le nom d'une variable unique dans le système. Dans FIP, ce sont les objets qui sont désignés et non les processus application, c'est pourquoi on peut considérer FIP comme un système de mise à jour et de gestion d'une base de données.

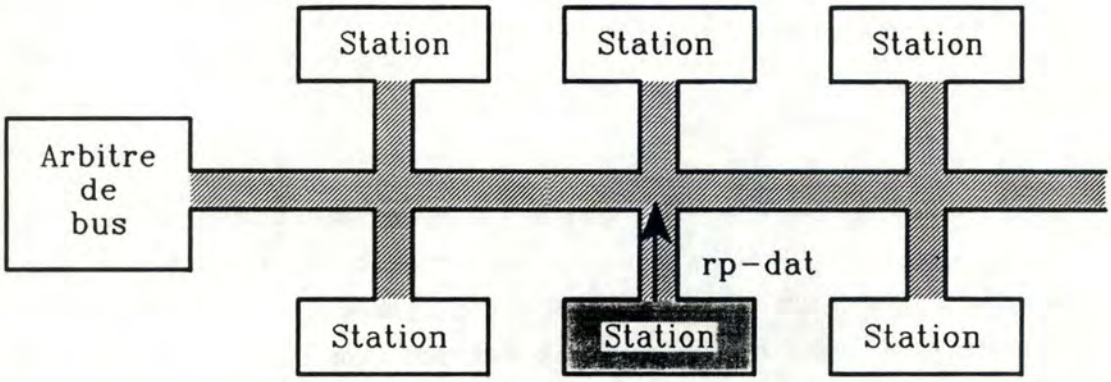
Toute transmission se fait sous la forme de deux trames successives :

- une trame identificateur (*id_dat*) : cette trame relève d'une fonction appelée arbitre de bus et permet de contrôler le droit d'accès des stations au médium.
- une trame de données (*rp_dat*) contenant la valeur de l'objet identifié dans la trame de nom.

Une transaction complète peut être décrite par quatre phases successives :

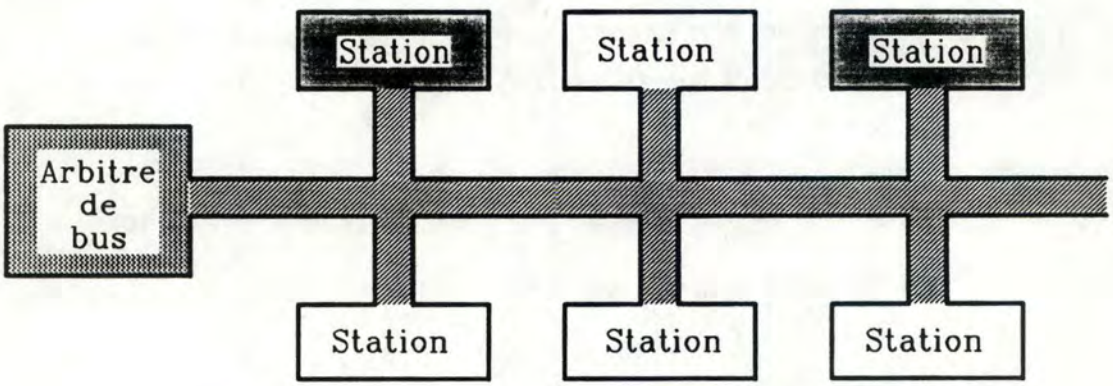
1. L'arbitre de bus émet sur le médium une trame identificateur *id_dat* (Cf. figure 6.7.).

2. Les stations productrices et consommatrices se reconnaissent respectivement comme productrices et consommatrices de la valeur de la variable associée à cet identificateur (Cf. figure 6.8.).
   On supposera ici que la station productrice de la valeur associée à l'identificateur est unique.



Phase 3 : l'unique station productrice envoie l'rp-dat

Figure 6.9.



Phase 4 : l'arbitre et les stations consommatrices consomment la valeur de l'rp-dat

Figure 6.10.

3. La station qui s'est reconnue comme productrice de la variable émet sur le médium une trame réponse *rp_dat* contenant la valeur de la variable associée à l'identificateur émis par l'arbitre de bus (Cf. figure 6.9.).
4. Les stations qui se sont reconnues comme consommatrices lisent la trame réponse émise sur le médium par la station productrice. L'arbitre de bus consomme également cette valeur afin de vérifier que l'opération s'est déroulée correctement (Cf. figure 6.10.).

. Service périodique et apériodique

La nomenclature des objets à transmettre est construite à partir des spécifications de l'application : on connaît quels sont les objets qui sont produits et ont besoin d'être échangés. De plus, la nomenclature se répète cycliquement et confère ainsi au bus FIP la fonction principale de *scrutation périodique d'états*.

Outre cela, FIP offre des mécanismes permettant d'assurer d'autres trafics et notamment le *trafic apériodique* d'objets identifiés non soumis au trafic périodique. Les besoins sont par exemple la transmission d'un événement, la retransmission d'objets en provenance du trafic périodique mais pour lesquels une erreur a été détectée.

L'objectif principal de FIP étant de supporter un échange périodique de données, la suite de ce travail sera consacrée à l'analyse de la couche liaison de données et plus particulièrement aux services périodiques sous-jacents.

. Services offerts par la couche liaison de données à la couche application.

La couche application demande l'exécution d'une certaine tâche à la couche liaison par une primitive *<primitive>.dem*. La couche liaison répond à une telle demande avec la primitive *<primitive>.conf*. Outre cela, la couche liaison informe la couche application d'une action qu'elle a effectuée sur le bus par une primitive *<primitive>.ind*.

. *L_PUT.dem(ID,VAL)*	: la couche application fait une demande de mise-à-jour de l'identificateur *ID*.
. *L_PUT.conf(ID)*	: la couche liaison de données confirme la mise-à-jour du buffer.
. *L_GET.dem(ID)*	: la couche application demande la valeur de l'identificateur *ID*.
. *L_GET.conf(ID,VAL)*	: la couche liaison de données fournit à la couche application la valeur *VAL* de l'identificateur *ID*.

. *L_SENT.ind(ID,VAL)* : la couche liaison de données signale à la couche application que la valeur *VAL* associée à l'identificateur *ID* vient d'être émise sur le bus.

. *L_RECEIVED.ind(ID,VAL)* : la couche liaison de données signale à la couche application que la valeur associée à l'identificateur *ID* vient d'être lue sur le bus.

3° La couche physique

La couche physique fournit aux stations la capacité de communiquer sur le bus FIP.

Les services offerts par la couche physique permettent l'émission et la réception de symboles de niveau liaison de données. Ces services fournissent le moyen aux entités liaison de données de coordonner leur émission et d'échanger des informations via le partage du médium.

6.3. Spécification formelle et analyse du protocole de scrutation cyclique du réseau FIP au moyen des réseaux de Petri temporels

Cette section reprend et complète une partie des travaux réalisés dans [MAR 89].

Ici, nous modélisons et analysons le protocole de liaison de données relatif à la scrutation cyclique de variables (service périodique) de FIP.

Dans un premier temps, nous donnons les contraintes temporelles du protocole à satisfaire et essayons d'établir des relations entre ces contraintes. Puis, nous effectuons la modélisation et la vérification du protocole à l'aide de l'outil RdP que nous avons présenté au chapitre 4. Enfin, nous concluons en montrant quel a été l'apport principal de la méthode basée sur le modèle des réseaux de Petri temporels, de même que l'apport apporté par l'utilisation de l'outil RdP.

6.3.1. Hypothèses simplificatrices

Le modèle de FIP étant de taille assez conséquente, plutôt que de le modéliser dans son entièreté, ce qui rendrait l'analyse inintelligible, nous nous sommes limités à la structure suivante :

- *un arbitre* dont le rôle est de gérer et de surveiller le trafic sur le bus,
- *une station* qui se reconnaît comme *productrice* de l'information,
- *une station* qui se reconnaît comme *consommatrice* de l'information,
- *un médium* constituant le lien entre l'arbitre, les stations productrice et consommatrice.

Ce choix simplifie la représentation des stations actives et est suffisant pour modéliser les temporisations relatives au protocole de scrutation cyclique (service périodique).

6.3.2. Les contraintes temporelles

A) Présentation et fonctions des contraintes temporelles

La définition des temporisations est impérative pour assurer l'interopérabilité des différents éléments connectés à un segment réseau FIP.

Le protocole que nous nous proposons d'étudier dans cette section est basé sur le respect de contraintes temporelles entre les entités arbitre, producteur et consommateur.

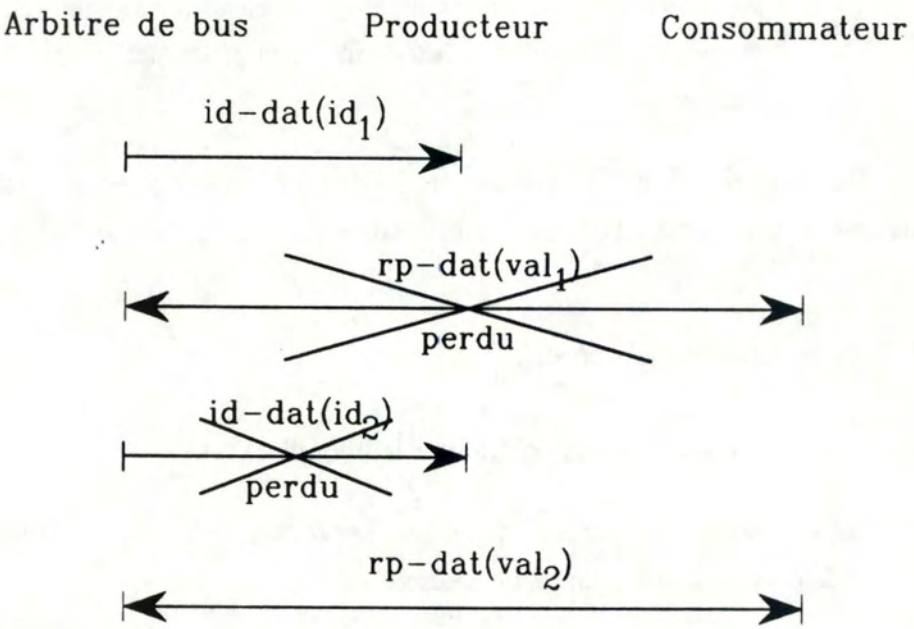


Figure 6.11. Cas de figure que la temporisation t4 permet d'éviter

Parmi les temporisations mentionnées dans [FIP 89] nous retiendrons celles qui sont en rapport direct avec l'objet de notre travail, c'est-à-dire celles relatives à la scrutation cyclique de variables.

Nous avons retenu quatre temporisations. Les deux premières représentent des *temps-limites*, tandis que les deux suivantes sont des *contraintes physiques du système*.

Nous allons étudier le bon fonctionnement de ces temps-limites en fonction des caractéristiques du système et de ses contraintes physiques.

1. La temporisation t1

- Elle est localisée dans l'arbitre de bus actif et elle a pour rôle de surveiller l'émission d'une trame réponse variable (*rp_dat*) dans un temps imparti.
- Elle est active lorsque l'arbitre de bus est dans l'état d'attente d'une trame réponse variable (*rp_dat*), suite à son émission respective d'une trame identificateur variable (*id_dat*).
- Quand la temporisation vient à expiration, l'arbitre de bus passe dans l'état d'émission de la trame identificateur suivante.

2. La temporisation t4

- Elle est localisée dans les stations consommatrices.
- Elle a pour rôle de surveiller l'émission sur le bus d'une trame réponse variable (*rp_dat*) dans un temps imparti.

  Insistons bien sur le fait que t4 ne vise pas uniquement la détection d'absence d'une trame réponse après une trame identificateur, mais qu'elle sert surtout dans le cas de figure représenté par la figure 6.11.

  t4 permet d'éviter que le consommateur ne consomme $rp\_dat(val_2)$ sur perte de deux trames successives $rp\_dat(val_1)$ et $id\_dat(id_2)$. En l'absence de la temporisation t4 ou avec une temporisation mal définie, le consommateur associerait la trame réponse variable $rp\_dat(val_2)$ à la trame identificateur variable $id\_dat(id_1)$, ce qui serait tout à fait erroné. L'expiration de t4 doit empêcher un tel phénomène en remettant le consommateur dans l'état d'attente de la trame identificateur suivante.

  t4 permet donc de bien associer à la trame identificateur reçue la trame réponse correspondante.

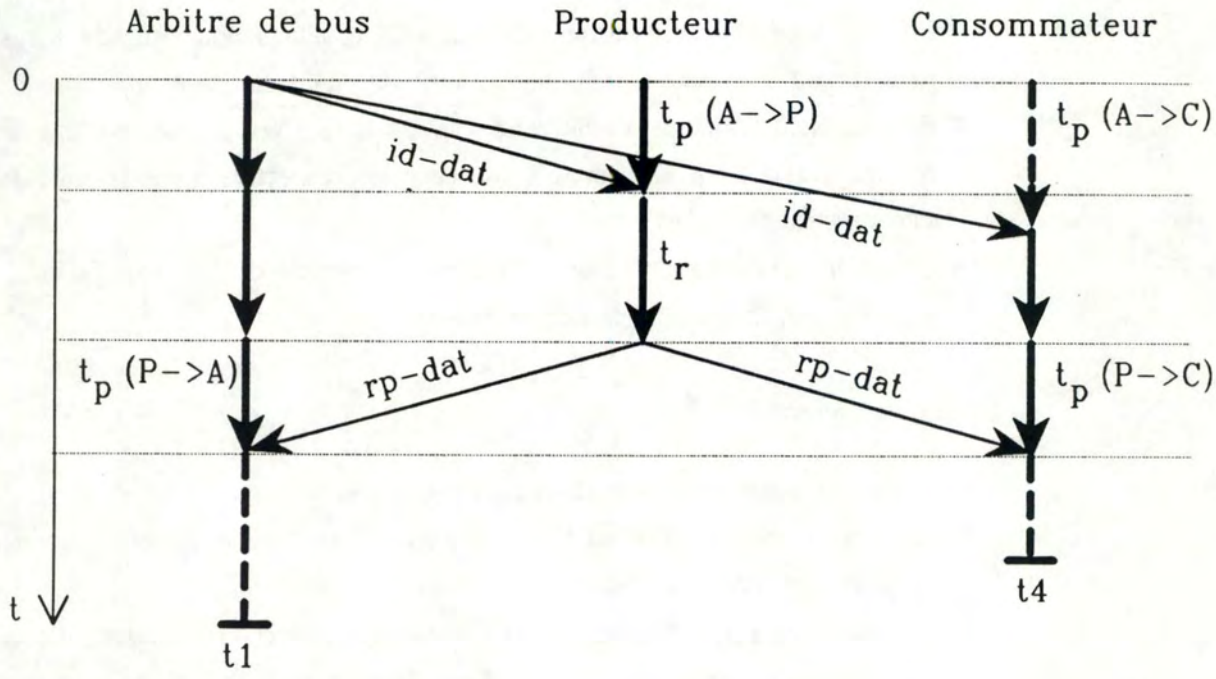


Figure 6.12. Diagramme d'échanges

3. Le temps de retournement t_r

- Le temps de retournement est localisé au niveau des stations et correspond au temps que met la station pour passer de l'état récepteur (réception d'une trame identificateur) à l'état émetteur (émission d'une trame réponse).

4. Le temps de propagation t_p

- Le temps de propagation dépend, d'une part, de la vitesse de propagation totale sur le bus et, d'autre part, de la position des différentes stations sur le bus.

B) Relations entre les différentes contraintes temporelles

Aucune valeur de temporisation ne nous étant donnée, nous avons jugé utile de déterminer des relations entre celles-ci afin d'éviter au maximum de devoir leur attribuer des valeurs au hasard pour pouvoir procéder à l'analyse.

D'une manière générale, les contraintes que doivent vérifier les différentes variables temporelles découlent de la définition des temporisations [FIP 89] et du diagramme des échanges représenté par la figure 6.12. et qui résume le fonctionnement du système.

A partir de là, il nous est possible d'établir les relations suivantes :

- $t1 > t_r + t_p(A \rightarrow P) + t_p(P \rightarrow A)$

  comme $t_p(A \rightarrow P) = t_p(P \rightarrow A)$ on peut écrire :

  $t1 > t_r + 2\, t_p(A \rightarrow P)$

- $t4 > t_r + t_p(A \rightarrow P) + t_p(P \rightarrow C) - t_p(A \rightarrow C)$

  si $t_p(A \rightarrow P) = t_p(P \rightarrow C)$ on peut écrire :

  $t4 > t_r + 2\, t_p(A \rightarrow P) - t_p(A \rightarrow C)$

Remarquons que dans la deuxième relation, $-t_p(A \rightarrow C)$ provient du fait que t4 ne prend cours qu'à partir du moment où le consommateur a reçu l'*id_dat*. Or, entre le moment où l'*id_dat* est envoyé par l'arbitre de bus (temps 0) et celui où le consommateur le reçoit, il s'écoule un certain temps. Ce temps, égal à $t_p(A \rightarrow C)$, logiquement, ne peut pas intervenir dans le calcul de t4.

De même, toujours dans la deuxième relation, il est essentiel de faire intervenir $t_p(A \rightarrow P)$ dans la détermination de la valeur de t4. Car, par définition, t4 a pour rôle de surveiller l'émission sur le bus d'une trame réponse variable, en l'occurrence *rp_dat*, dans un temps imparti.

Or, pour que le producteur envoie ce *rp_dat*, il faut qu'il ait reçu l'*id_dat* de l'arbitre et il ne le reçoit qu'après un temps égal à $t_p(A \rightarrow P)$. Il est donc impossible que le consommateur capte l'émission d'un *rp_dat* avant le temps $t_p(A \rightarrow P)$. Pour s'en convaincre, il suffit de considérer $t_p(A \rightarrow P)$ supérieur à $t_p(A \rightarrow C)$. Après un temps égal à $t_p(A \rightarrow C)$, le consommateur reçoit un *id_dat* et passe dans l'état d'attente du *rp_dat* correspondant. Ce *rp_dat* ne sera transmis sur la ligne que quand une station se sera reconnue comme productrice, c'est-à-dire quand l'*id_dat* sera parvenue jusqu'à elle, soit dans ce cas, après $t_p(A \rightarrow P)$. Le consommateur, avant de pouvoir capter l'*rp_dat*, devra donc attendre au moins un temps égal à $t_p(A \rightarrow P) - t_p(A \rightarrow C)$.

C) Interprétation des relations

Des deux relations ci-dessus, nous déduisons facilement que $t1 \geq t4$, ce qui exprime le fait que le consommateur a dû consommer et être en position de pouvoir être de nouveau sensibilisé quand un nouvel *id_dat* arrive à lui.

Ces relations soulignent également l'importance du temps de retournement du producteur, du temps de propagation sur le bus et des contraintes physiques en général, dans l'expression des temporisations correspondant à des temps-limites.

Dès lors, les valeurs des contraintes de temporisations peuvent être influencées par la position des stations sur le bus.

Dans [MAR 89], on trouve trois cas de configuration possible :

1. Arbitre et consommateur aux extrémités et producteur au milieu
   (configuration A-P-C)

$$t1 > t_r + t_p$$
$$t4 > t_r$$

2. Arbitre et producteur aux extrémités et consommateur au milieu
   (configuration A-C-P)

$$t1 > t_r + 2\,t_p$$
$$t4 > t_r + t_p$$

3. Producteur et consommateur aux extrémités et arbitre au milieu
   (configuration P-A-C)

$$t1 > t_r + t_p$$
$$t4 > t_r + t_p$$

De manière générale, pour que les relations soient correctes dans tous les cas de figures, on aura :

$$t1 > t_r + 2\,t_p$$
$$t4 > t_r + t_p$$
$$t1 \geq t4$$

6.3.3. Le modèle des réseaux de Petri temporels

A) Extension du modèle des réseaux de Petri temporels

Pour modéliser le protocole, nous étendons le formalisme des réseaux de Petri temporels au modèle des réseaux de Petri temporels étiquetés, tel que nous l'avons défini au chapitre 5 et qui consiste à associer à chaque transition communicante une étiquette de la forme ?X/!Y. Pour rappel, ?X représente la réception du message X et !Y l'émission du message Y.

B) Découpe en modules

Dans une première étape, nous avons choisi de modéliser séparément le comportement des différentes entités de protocole (arbitre de bus, producteur, consommateur) et du milieu de transmission (médium).

Il nous restera alors à interconnecter les différents modules obtenus pour obtenir le modèle global.

Cette découpe en modules permet non seulement de réduire l'effort de représentation mais est aussi parfaitement bien adaptée à l'utilisation de l'outil RdP tel que présenté au chapitre 4.

1° L'arbitre de bus

. Temporisation t1

La temporisation t1 empêche l'arbitre d'attendre indéfiniment l'*rp_dat* correspondant à l'*id_dat* qu'il a envoyé.

Dans le modèle des réseaux de Petri temporels, nous spécifions la temporisation t1 par l'intervalle $[t1_a, t1_b]$, ce qui permet de prendre en compte les fluctuations éventuelles de l'horloge.

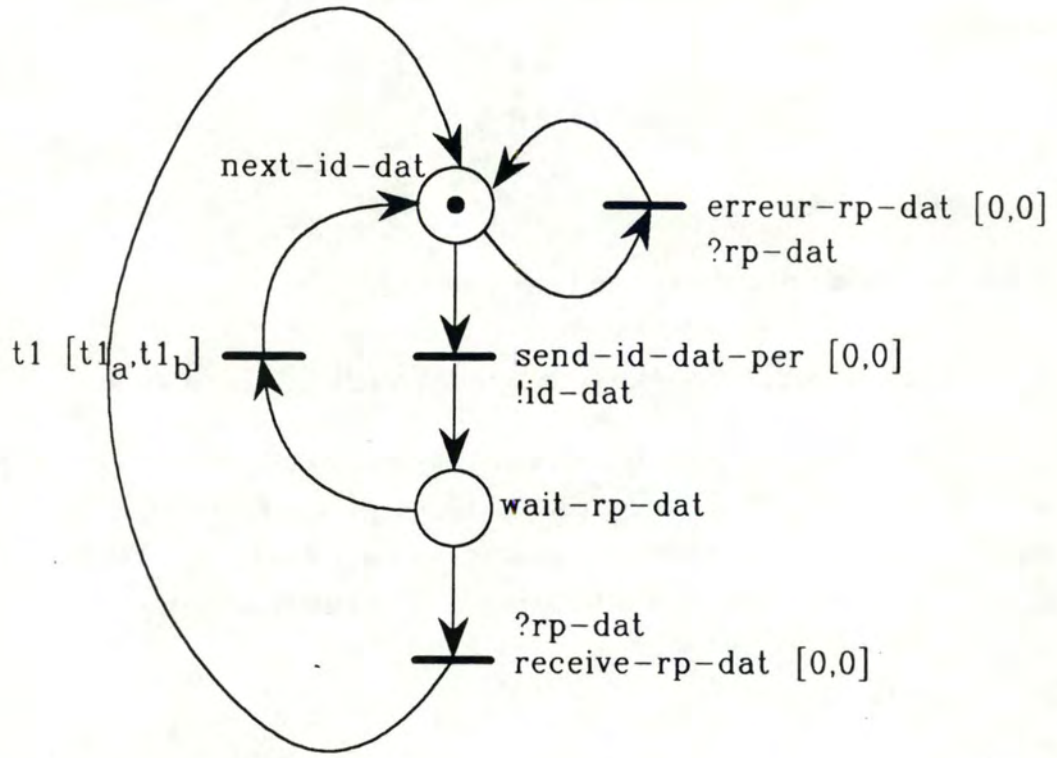


Figure 6.13. Modèle de l'arbitre

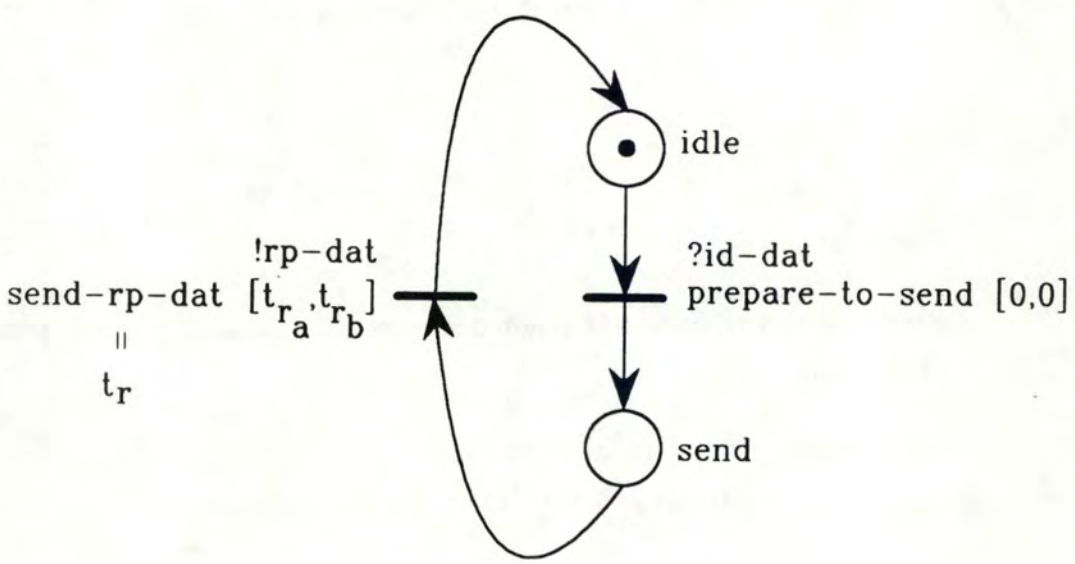


Figure 6.14. Modèle du producteur

. Situations de mauvais fonctionnement dues au mauvais calcul du temps-limite

La réception d'un *rp_dat* alors que l'arbitre se trouve dans l'état prêt à émettre un *id_dat* (état *next-id-dat*) caractérise un mauvais fonctionnement du système dû à un temps-limite mal établi.

. Modèle (Cf. figure 6.13.)

La place *next_id_dat* représente l'état dans lequel l'arbitre de bus est prêt à émettre un *id_dat*. Si, dans cet état, l'arbitre de bus reçoit un *rp_dat*, il y a erreur.

Lorsque l'*id_dat* est envoyé, l'arbitre de bus passe dans l'état d'attente d'un *rp_dat* (état *wait_rp_dat*). Dans cet état, deux situations peuvent se produire :

- soit l'arbitre de bus reçoit l'*rp_dat* avant l'expiration de la temporisation t1. Dans ce cas, tout va bien et l'arbitre de bus revient dans l'état prêt à émettre l'*id_dat* suivant (place *next_id_dat*).
- soit la temporisation t1 expire et on revient également dans l'état *next_id_dat*.

Ceci modélise bien le fait qu'une fois que la trame correspondant à l'*id_dat* a été envoyée, l'*rp_dat* doit arriver dans le temps imparti par la temporisation t1; si ce n'est pas le cas, on considère qu'il est définitivement perdu. Si cette temporisation est mal calculée, l'*rp_dat* arrive après expiration de la temporisation, quand l'arbitre est passé dans l'état *next_id_dat*. Il y a alors erreur (transition *erreur_rp_dat*).

2° Le producteur

. Temporisation t_r

t_r représente le temps de retournement de la station productrice, c'est-à-dire le temps qui s'écoule pour que le modem passe de l'état d'émetteur à l'état de récepteur et inversement.

Pour prendre en compte ses variations, nous considérons que le temps de retournement correspond à une valeur comprise dans l'intervalle $[t_{r.a}, t_{r.b}]$.

. Modèle (Cf. figure 6.14.)

Dans l'état *idle*, le producteur est en attente d'un *id_dat*. A la réception de l'*id_dat* (*?id_dat*), le producteur passe dans l'état *send* dans lequel il est prêt à envoyer l'*rp_dat* correspondant à l'*id_dat* reçu. Après le temps de retournement du modem, correspondant à l'intervalle de temps associé à la transition *send_rp_dat*, le producteur envoie l'*rp_dat* (*!rp_dat*) et revient à l'état *idle*.

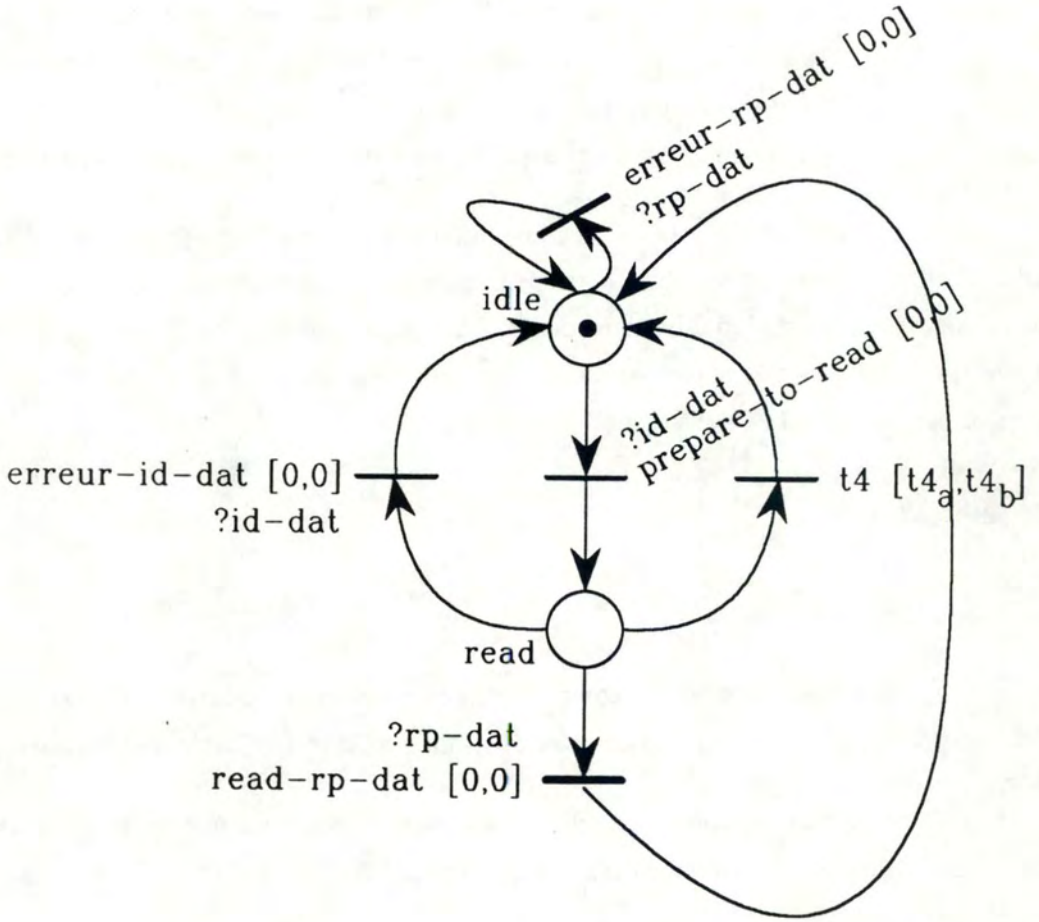


Figure 6.15. Modèle du consommateur

3° Le consommateur

. Temporisation t4

Le but de la temporisation t4 est que le consommateur ne reste pas indéfiniment dans l'état d'attente du *rp_dat* associé à l'*id_dat* qu'il a préalablement reçu. Il attendra au maximum le temps que dure la temporisation.

Comme ce temps-limite d'attente peut subir des variations (par exemple, variations dues à celles de l'horloge), nous le définissons par l'intervalle $[t4_a, t4_b]$.

. Situations de mauvais fonctionnement dues au mauvais calcul du temps-limite

La réception d'un *rp_dat*, ou encore d'un *id_dat*, alors que le consommateur est dans l'état d'attente d'un *id_dat*, respectivement d'un *rp_dat*, sont deux situations de mauvais fonctionnement du système qui peuvent se produire si la valeur du temps-limite est mal établie.

. Modèle (Cf. figure 6.15.)

Dans l'état *idle*, le consommateur attend un *id_dat*. Si au lieu de recevoir un *id_dat*, il reçoit un *rp_dat* (*?rp_dat*), il y a erreur (transition *erreur_rp_dat*). Sinon, s'il y a réception d'un *id_dat* (*?id_dat*), le consommateur passe de l'état *idle* à l'état *read* où il est prêt à lire la valeur de l'*rp_dat* correspondant à l'i*d_dat* qu'il vient de recevoir. Trois cas d'évolution sont alors possibles, tous trois menant à l'état initial :

- soit un *id_dat* arrive, ce qui constitue une erreur (transition *erreur_id_dat*),
- soit le temps d'attente imparti de l'*rp_dat* est écoulé,
- soit il y a réception de l'*rp_dat*.

4° Le médium

. Temporisation t_p

t_p correspond au temps de propagation sur la ligne.

. Modèle

Le médium est ici un système de transmission de type half-duplex.

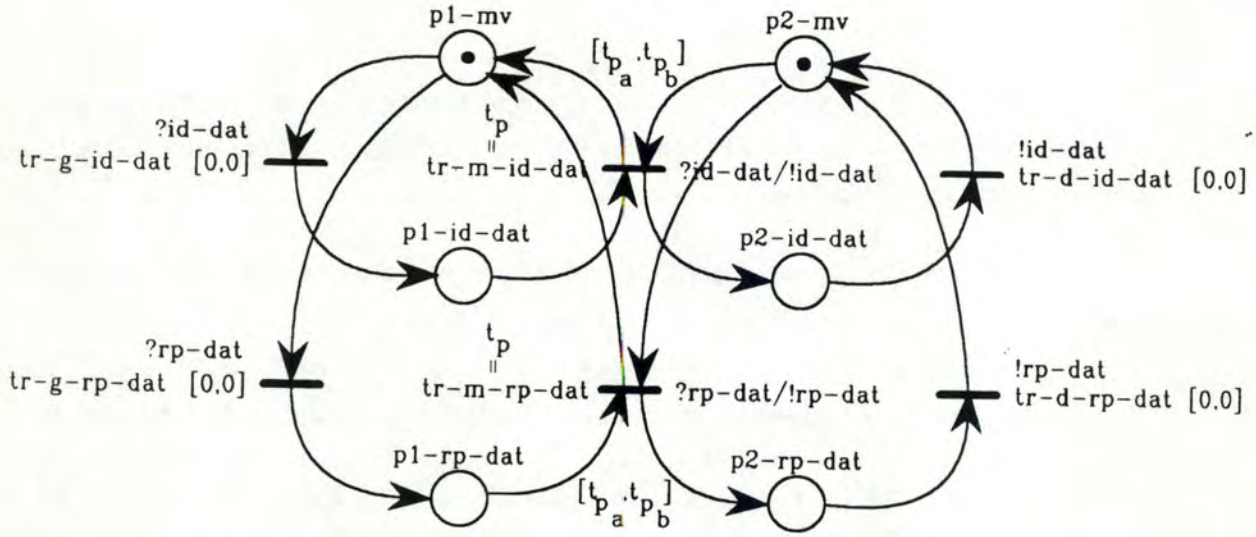


Figure 6.16. Modèle du médium
(parcours des trames id-dat et rp-dat dans le même sens)

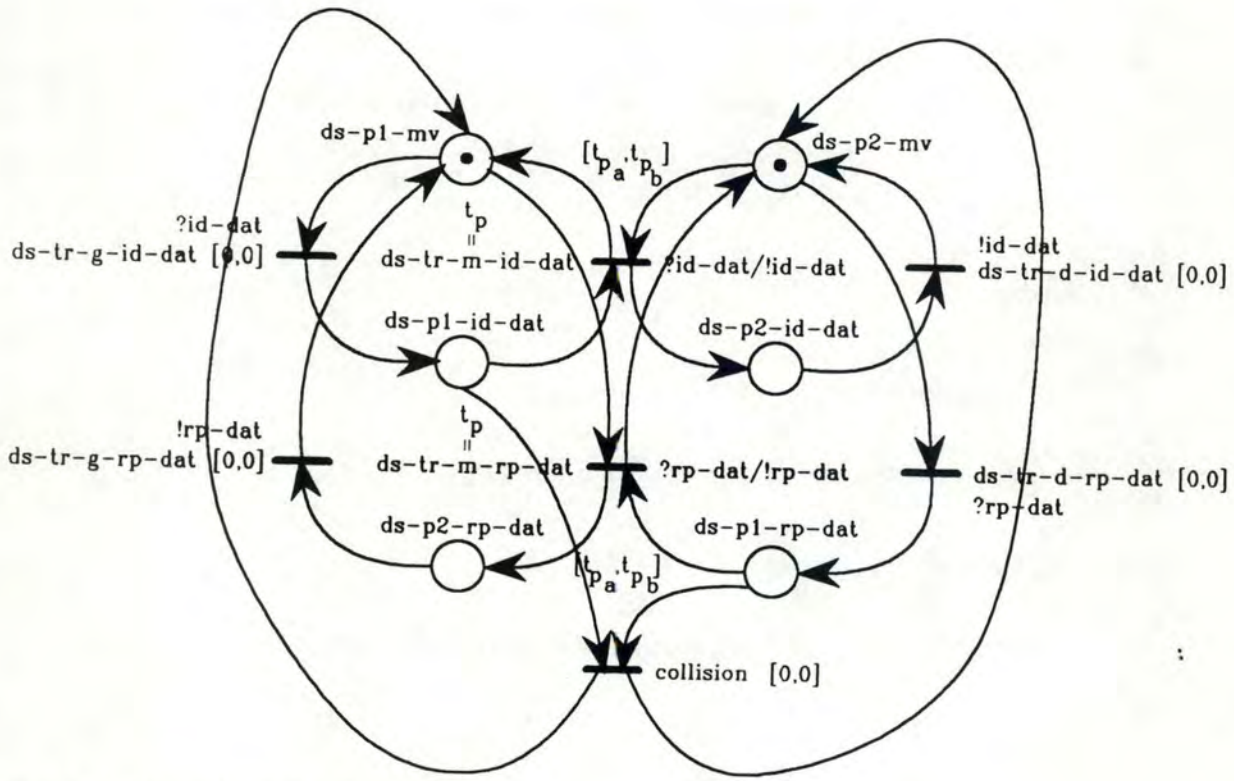


Figure 6.17. Modèle du médium
(parcours des trames id-dat et rp-dat dans le même sens)

Deux types de messages peuvent circuler sur ce médium :

- *id_dat*
- *rp_dat*

Ces trames peuvent entrer en collision.

C'est pourquoi, nous devons faire une double modélisation :

- un modèle présentant deux types de messages pouvant circuler sur le médium (parcours des trames *id_dat* et *rp_dat* dans le même sens).
- un modèle représentant les collisions qui peuvent se produire lorsque deux trames circulent dans des directions opposées (parcours des trames *id_dat* et *rp_dat* en sens inverse).

Premier modèle : parcours des trames id_dat et rp_dat dans le même sens
(Cf. figure 6.16.)

Les trames *id_dat* et *rp_dat* sont véhiculées de la gauche vers la droite de la figure mais, pas simultanément. C'est pourquoi, la place *p1_mv* est initialisée avec un seul jeton.

La place *p1_id_dat* (*p1_rp_dat*) représente l'introduction d'un *id_dat* (respectivement d'un *rp_dat*) sur le médium. Pour que cela soit possible, il faut que le médium soit vide, ce que l'on représente par la présence d'un jeton dans la place *p1_mv*. Après un certain temps, représentant le temps de propagation dans le médium et modélisé par l'intervalle $[t_{p.a},t_{p.b}]$ de la transition *tr_m_id_dat* (respectivement *tr_m_rp_dat*), la trame est véhiculée vers la sortie du médium. Ceci ne peut se produire que si la sortie est libre, ce qui est caractérisé par la présence d'un jeton dans la place *p2_mv*.

Deuxième modèle : parcours des trames id_dat et rp_dat en sens inverse
(Cf. figure 6.17.)

Le comportement de ce modèle est semblable à celui précédemment décrit, en considérant, toutefois, que les trames *id_dat* sont véhiculées de la gauche vers la droite de la figure, tandis que les trames *rp_dat* sont véhiculées de la droite vers la gauche.

Notons également, que s'il y a introduction à la fois d'une trame *id_dat* (place *ds_p1_id_dat*) et d'une trame *rp_dat* (place *ds_p1_rp_dat*), comme les deux trames circulent en sens inverse, il y a collision (Cf. transition *collision*).

C) Le modèle global (Cf. figure 6.18.)

Le modèle global est obtenu en interconnectant les modèles locaux des différentes stations et du médium.

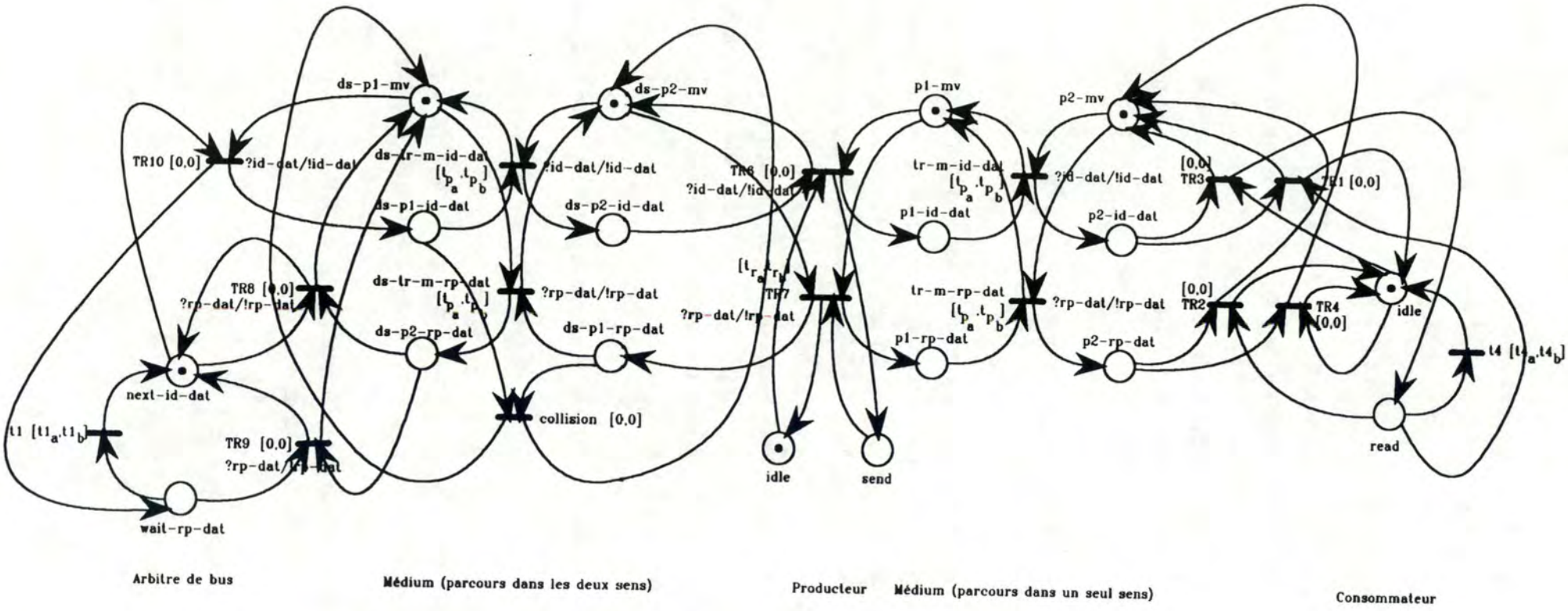


Figure 6.18. Modèle global

Pour la modélisation globale, nous avons vu que plusieurs configurations étaient possibles. Nous avons choisi d'analyser ici la configuration A-P-C, qui s'est avérée meilleure que les autres, du moins pour les valeurs des temporisations que nous nous étions fixées au départ de l'analyse.

L'interconnexion (communication) entre les différents modèles peut être obtenue de deux manières différentes :

- par *places partagées*
- par *rendez-vous* (synchronisation forte de transitions)

. Communication par places partagées

Ainsi que nous l'avons vu précédemment, l'interconnexion de deux modèles selon cette technique n'est possible que si une ou plusieurs transitions des deux modèles distincts référencent une même place. Cette place est alors locale à un modèle et importée dans l'autre.

Dès lors, dans le cas qui nous intéresse, cette démarche nécessiterait de remplacer, dans les différents modèles, les étiquettes ?X/!Y associées aux transitions de la manière suivante :

- ?X serait remplacée par une place d'entrée supplémentaire.
- !Y serait remplacée par une place de sortie supplémentaire.

Mentionnons également le fait que le partage des places entre modèles contraint le parallélisme et peut nuire à la compréhension du modèle, limitant ainsi les possibilités de vérification.

Nous préférons donc utiliser le mécanisme de communication par rendez-vous entre modèles, qui, outre le fait d'éviter ces inconvénients, permet d'obtenir un graphe des classes d'états du modèle global de taille inférieure à celui obtenu avec le mécanisme des places partagées.

. Communication par rendez-vous

Pour rappel, le mécanisme de rendez-vous consiste à fusionner les transitions communicantes des différents modèles en une transition commune. La transition résultante est constituée de la réunion des préconditions et de la réunion des postconditions des différentes transitions d'origine. Son étiquette résulte du produit des deux étiquettes d'origine.

On dit qu'il y a *synchronisation forte* entre les modèles car le rendez-vous correspond à considérer comme atomique l'émission et la réception de la trame.

6.3.4. Modélisation à l'aide de l'outil RdP

Pour une meilleure compréhension de ce qui va suivre, nous conseillons vivement au lecteur de se référer aux différentes vues présentes dans le document RdP se trouvant en annexe 2 et qui représentent les étapes de l'implémentation sous RdP des modèles construits ci-dessus.

Le système global, désigné sous le vocable *fip_per*, est vu comme une architecture de cinq sous-systèmes communiquant.

A chaque sous-système est associé un objet bloc (Cf. page 1 du document RdP) :

- arbitre,
- mhdds (médium half-duplex avec parcours des trames dans les deux sens),
- producteur,
- mhdms (médium half-duplex avec parcours des trames dans le même sens),
- consommateur.

Au niveau le plus bas, le comportement de chaque sous-système est décrit par un réseau de Petri indépendant :

- pn arbitre,
- pn mhdds,
- pn producteur,
- pn mhdms,
- pn consommateur.

Chacun de ces réseaux de Petri a la forme du modèle correspondant, décrit à la section précédente.

Les communications entre les cinq sous-systèmes sont exprimées par des canaux bi-directionnels reliant les différents sous-systèmes de la manière qui suit (Cf. page 2 du document RdP) :

- le sous-système arbitre est relié au sous-système mhdds par le canal bi-directionnel *phy*.
- le sous-système mhdds est relié au sous-système mhdms par le canal bi-directionnel *inter-med* et au sous-système producteur par le canal *phyp-mhdds*.
- le sous-système mhdms est relié au sous-système producteur par le canal *phyp-mhdms* et au sous-système consommateur par le canal *phyc*.

Chaque réseau de Petri temporel décrivant un des cinq sous-systèmes est relié à son bloc englobant par un ou plusieurs canaux obligatoirement uni-directionnel (Cf. pages 3, 6, 10, 13 et 17 du document RdP) :

- le réseau de Petri modélisant le comportement du sous-système arbitre se connecte au canal de niveau supérieur *phy* par le canal sortant *output-phy* et par le canal entrant *input-phy* (page 3).
- ... et ainsi de suite pour les différents réseaux de Petri modélisant les sous-système mhdds (page 6), producteur (page 10), mhdms (page 13) et consommateur (page 17).

Les échanges de messages entre les différents réseaux de Petri modélisés se font par ces canaux entrant et sortant, en passant par les canaux de niveau supérieur reliant les blocs des différents sous-systèmes entre eux.

Pour que ces échanges puissent se produire, il faut encore définir les communications au niveau de la représentation sous forme des réseaux de Petri.

Ainsi que nous l'avons vu, la définition des communications peut se faire de deux manières : par *rendez-vous* (synchronisation de transitions) ou par *partage de places*.

Pour des raisons déjà évoquées, nous avons opté pour la communication par synchronisation de transitions. Dans RdP, celle-ci s'effectue en deux temps :

1. Après création d'une transition dans le module réseau de Petri temporel (la transition est dite alors locale), la transition est déclarée comme émettrice d'un message Y par l'intermédiaire d'un canal sortant de ce réseau et figurant en interface si il existe. Elle prend alors le nom *output* *<nom_canal_émetteur>!<nom_message>*, ce qui signifie, sur le canal émetteur <nom_canal_émetteur> émission du message <nom_message>.
   La commande *output on* permet de réaliser cela en associant le caractère émetteur à la transition (symbole ).

2. Dans tous les autres réseaux de Petri pour lesquels il existe un canal entrant relié (par une hiérarchie de canaux) à ce canal émetteur, la résolution de la synchronisation est proposée par l'intermédiaire d'une transition du type *input* *<nom_canal_récepteur>?<nom_message>* proposée à l'interface; ce qui signifie, réception sur le canal <nom_canal_récepteur> du message <nom_message>. La résolution de la synchronisation s'effectue à l'aide de la commande *input on* qui associe le caractère récepteur à la transition (symbole ).

6.3.5. Analyse des temporisations et résultats

Pour analyser les temporisations du protocole de scrutation cyclique (service périodique), nous utilisons l'analyseur de réseaux de Petri temporels RdPt.

Avant de commencer l'analyse proprement dite, établissons un certain nombre de propriétés que notre modèle doit respecter pour pouvoir présenter un fonctionnement correct. Ces propriétés sont les suivantes :

- borné (condition suffisante pour que le graphe des classes d'états soit fini).
- cyclique, c'est-à-dire réinitialisable.
- sans blocage (le graphe des classes d'états ne doit contenir aucune classe menant à une impasse).
- non vivant car les transitions modélisant des situations d'erreurs ne peuvent être tirées dans un réseau dont le comportement est normal.

A présent, nous allons étudier différentes temporisations et pour chaque modèle résultant nous poursuivrons l'analyse en deux étapes.

Dans une première étape, nous nous assurerons que le modèle vérifie bien les propriétés mentionnées ci-dessus.

Dans une seconde étape, le graphe des classes d'états nous permettra de vérifier si le fonctionnement du réseau est correct et s'il n'y a pas de mauvais fonctionnements dus à un mauvais calcul des temporisations (collision, réception d'un *id_dat* alors que la station attend un *rp_dat* et inversement).

Nous vérifierons également que la valeur des temporisations t1 et t4 (temps-limites) est effectivement fonction des contraintes physiques du système, telles que le temps de retournement et le temps de propagation. Pour cela, nous ferons évoluer ces contraintes en étudiant leurs répercussions sur les temporisations t1 et t4.

Nous essayerons aussi de fixer les bornes inférieures et supérieures des intervalles des temporisations t1 et t4 de telle sorte qu'elles laissent le comportement du système sans erreur.

Pour commencer l'analyse, nous avons considérer les unités de temps suivantes pour les variables temporelles :

- t_p = 2; nous considérons également par la suite le temps de propagation dans la moitié du bus que nous noterons T ($T = t_p/2$).
- [t1]=[16,18] [t4]=[13,15] [T]=[1,1] [t_T]=[9,11]

Les valeurs de ces temporisations respectent les inéquations établies au point 6.3.2. paragraphe B de ce chapitre.

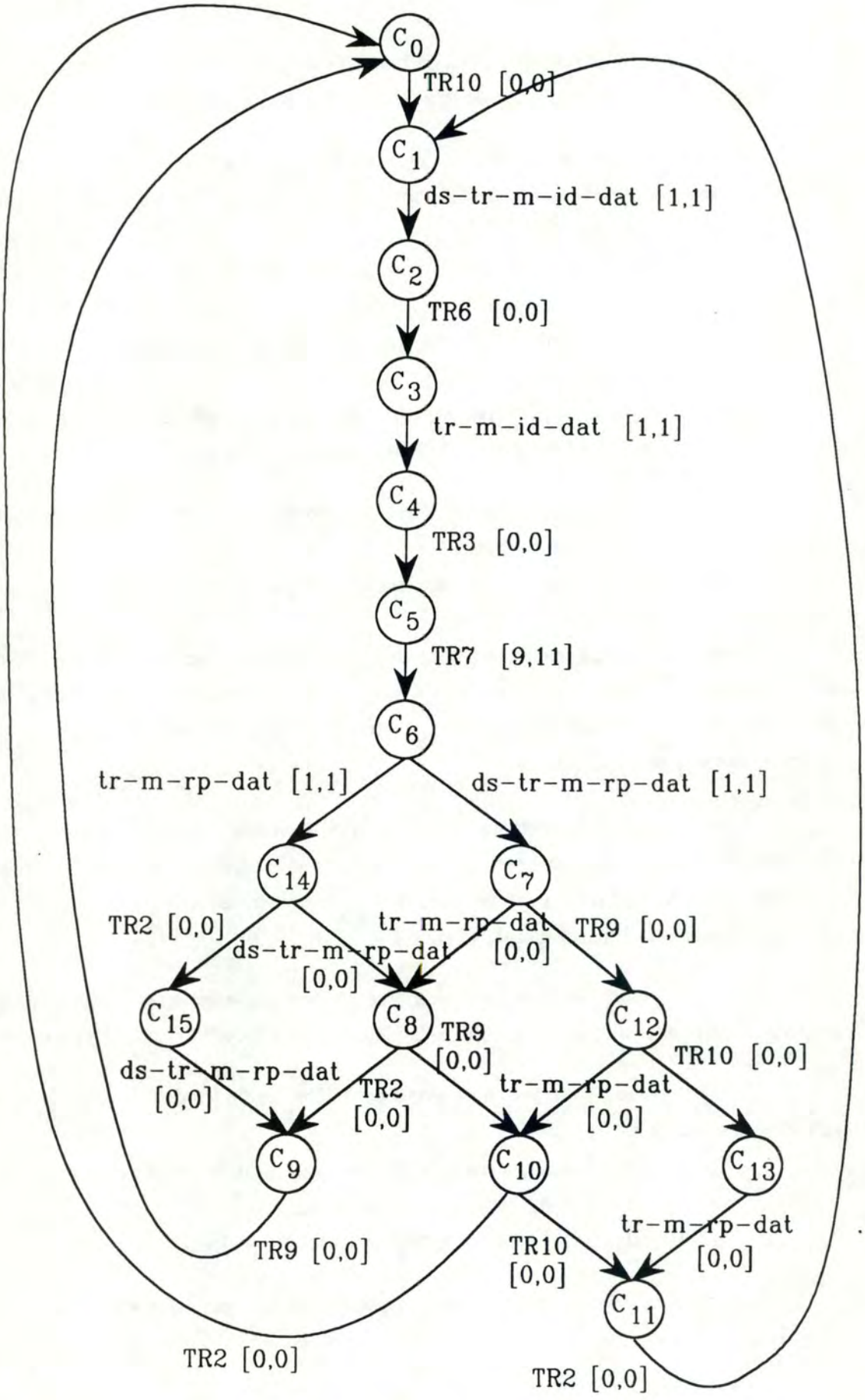


Figure 6.19. Graphe des classes d'états

L'analyse nous a permis de vérifier que le réseau produit à partir de ces variables temporelles est borné à 16 classes d'états, réinitialisable, non vivant et ne contient aucun blocage.

Le graphe des classes d'états que nous avons obtenu est représenté par la figure 6.19.

Légende :

. *TR2* est la transition résultante de la fusion des transitions *read_rp_dat* (consommateur) et *tr_d_rp_dat* (mhdms).

. *TR3* est la transition résultante de la fusion des transitions *prepare_to_read* (consommateur) et *tr_d_id_dat* (mhdms).

. *TR5* est la transition résultante de la fusion des transitions *send_rp_dat* (producteur) et *tr_g_rp_dat* (mhdms).

. *TR6* est le transition résultante de la fusion des transitions *prepare_to_send* (producteur), *tr_g_id_dat* (mhdms) et *ds_tr_d_id_dat* (mhdds).

. *TR7* est la transition résultante de la fusion des transitions *TR5* et *ds_tr_d_rp_dat* (mhdds).

. *TR9* est la transition résultante de la fusion des transitions *receive_rp_dat* (arbitre) et *ds_tr_g_rp_dat* (mhdds).

. *TR10* est la transition résultante de la fusion des transitions *send_id_dat_per* (arbitre) et *ds_tr_g_id_dat* (mhdds).

Le contenu des classes d'états, ainsi que les résultats de l'analyse produit par RdPt, se trouvent à l'annexe 3.

Notons que le graphe des classes d'états est cyclique et que nous avons une évolution purement séquentielle qui traduit un fonctionnement normal du système.

Nous avons poursuivi l'analyse en modifiant les bornes temporelles des temporisations.

Ceci nous a permis de déterminer quelles étaient les valeurs minimale et maximale que pouvaient prendre les différentes temporisations, ainsi que de vérifier les relations entre ces dernières.

A) Valeur minimale des bornes inférieures de [t1] et [t4]

Pour t_p=[1,1] et t_r=[9,11], l'intervalle [t1] admet une borne inférieure $\geq$ 15 et l'intervalle [t4] une borne inférieure $\geq$ 13. Toute temporisation inférieure produit des erreurs dans le fonctionnement du système.

Par exemple, si l'on considère les temporisation [t1]=[15,18] et [t4]=[12,15], le réseau est toujours non vivant, réinitialisable et sans blocage mais il est borné à 26 classes d'états au lieu de 16 (Cf. résultats de l'analyse figurant à l'annexe 4).

Si l'on étudie le graphe des classes d'états nous constatons qu'un cas d'erreur peut se présenter, à savoir la réception par le consommateur d'un *rp_dat* alors qu'il se trouve dans l'état d'attente d'un *id_dat*. Ceci est dû à la temporisation t4 qui est arrivée à expiration avant que l'*rp_dat* n'ait eu le temps d'atteindre le consommateur.

De même, si l'on considère les temporisation [t1]=[13,18] et [t4]=[13,15], le réseau est non vivant, réinitialisable, sans blocage et borné à 41 classes d'états. Le graphe des classes d'états montre alors deux situations erronées possibles : la collision d'un *id_dat* et d'un *rp_dat* sur le médium ou la réception d'un *rp_dat* par l'arbitre de bus alors qu'il est prêt à émettre un *id_dat*. Ces deux situations sont dues à une mauvaise estimation de la temporisation t4.

B) Valeur maximale des bornes supérieures de [t1] et [t4]

La valeur maximale des bornes supérieures des intervalles [t1] et [t4] peut prendre des valeurs infinies, à condition toutefois que la relation $t1 \geq t4$ soit vérifiée. Cependant, il est évident que pour l'efficacité et le bon déroulement du système, attribuer des bornes infinies n'a aucun sens, car en cas de non réception d'une trame, la ou les stations pourraient attendre indéfiniment. Les bornes supérieures doivent donc être fixées à des valeurs raisonnables.

C) Relations entre les différentes temporisations

. Relation entre t1 et t4

Quelles que soient les valeurs prises par les bornes des intervalles [t1] et [t4], celles-ci ne peuvent non seulement être inférieures à leur valeur minimale permise mais doivent aussi vérifier la relation $t1 \geq t4$. Dans le cas contraire, nous avons observé des erreurs de fonctionnement du réseau telles que collision, réception d'un *rp_dat* alors que la station attend un *id_dat*, ...

. Influence du temps de retournement sur les temporisations t1 et t4

Pour vérifier l'influence du temps de retournement sur les intervalles [t1] et [t4], nous avons fait fluctuer celui-ci vers le bas et vers le haut. Nous avons constaté que la valeur minimale de la borne inférieure de [t1] et [t4] évoluait proportionnellement aux variations du temps de retournement.

Ainsi, si $t_r=[7,9]$ (si $t_r=[11,13]$), la valeur minimale de la borne inférieure de [t1] devient 13 (respectivement 17) et la valeur minimale de la borne inférieure de [t4] devient 11 (respectivement 15).

Ce qui vient d'être dit ici confirme les relations que nous avions établies au point 6.3.2., y compris la relation $t1 \geq t4$. Ce résultat est d'autant plus intéressant que la norme de FIP

fournit une relation d'égalité entre ces deux temporisations t1 et t4. Un réajustement de cette relation peut donc être envisagé.

6.4. Conclusion

Nous avons utilisé une méthodologie pour la vérification des protocoles de communication, basée sur l'utilisation des réseaux de Petri temporels. La méthode par énumération des classes d'états a été appliquée au protocole de scrutation cyclique (couche liaison de données). Elle s'est avérée bien adaptée à la prise en compte des contraintes temporelles dans les systèmes distribués temps réel.

Des hypothèses simplificatrices sur le nombre de stations ont été introduites lors de la réalisation du modèle, afin de permettre la vérification formelle par la technique d'analyse énumérative des réseaux de Petri temporels. Cette technique présente cependant des limitations, en particulier le risque d'explosion du nombre de classes d'états. Si l'explosion combinatoire n'est pas contrôlée par une modélisation adaptée, le coût de l'analyse augmente en conséquence. D'où, la nécessité de réduire au maximum le nombre de stations et de limiter la modélisation aux interactions entre la couche liaison de données et le médium. Nous avons toutefois tenté de compléter notre modèle initial en y ajoutant les services offerts par la couche liaison de données à la couche application, mais le nombre de classes d'états était tel que nous avons dû très vite y renoncer.

L'analyse formelle de ce protocole a donné des résultats intéressants. Elle a conduit à considérer, entre les temporisations t1 et t4, non plus une relation d'égalité ainsi que spécifiée dans la norme FIP [FIP 89], mais plutôt une relation de la forme $t1 \geq t4$.

Nous avons pu ensuite déterminer quel était l'impact de mauvaises temporisations sur le fonctionnement du système, de même que de souligner l'importance de la position des stations sur le bus pour le calcul de ces mêmes temporisations.

L'analyse a également montré qu'il était possible de déterminer les valeurs minimales et maximales que peuvent prendre les différentes temporisations de telle sorte que la cohérence du système soit maintenue.

Elle a permis de vérifier des relations que nous avions préalablement établies entre les différentes temporisations et a précisé l'importance du temps de retournement d'une station et des contraintes physiques du système pour la définition des temporisations.

Notons toutefois que la méthode ne procure pas de relation entre les temporisations mais permet simplement de les vérifier a posteriori en prouvant le fonctionnement correct du protocole. On ne peut donc déterminer leurs valeurs admissibles directement. D'où l'importance d'essayer de déterminer au préalable ces relations afin d'éviter d'avancer à l'aveuglette.

Les résultats obtenus montrent l'importance d'analyser le comportement d'un protocole du type de celui que nous avons analysé ici, dans son environnement réel, avec la prise en compte du temps. Cependant, si le formalisme des réseaux de Petri temporels s'avère puissant, la méthodologie d'analyse développée part d'hypothèses simplificatrices (nombres de stations, ...) et

nous ne pouvons donc étendre sans restrictions, à un nombre arbitraire de stations, la preuve du comportement correct faite sur une configuration telle que nous l'avons modélisée.

Terminons ces quelques remarques en disant que l'application traitée dans ce chapitre a permis de tester l'outil RdP et son analyseur de réseaux de Petri temporels, RdPt. Le mode de représentation qu'il offre, sous forme de sous-systèmes communicants (architecture modulaire), ainsi que les fonctions d'analyse, sont particulièrement bien adaptés à la modélisation et à la vérification de réseaux de taille importante. L'énumération permet de produire des graphes conséquents mais qui restent toutefois lourds à manipuler. Incorporer à l'analyseur RdP des techniques de réduction du graphe des classes d'états, telles que celles mentionnées aux chapitre 4, ne serait donc pas un luxe pour cet outil qui n'en demeure pas moins très performant.

CONCLUSION

L'analyse de la littérature se rapportant au modèle des réseaux de Petri temporels nous avait permis de constater que ce sujet était délaissé depuis plusieurs années. Il semblerait que cela soit lié à la difficulté de maîtriser les paramètres temporels de ce modèle pour l'élaboration d'une méthode d'analyse systématique. Or, des méthodes d'analyse existent et certaines ont été développées au Laboratoire d'Automatique et d'Analyse des Systèmes de Toulouse.

C'est pourquoi, dans ce mémoire, nous avons tenu à étudier le modèle des réseaux de Petri temporels et une méthode d'analyse qui y est associée, afin de montrer leur utilité pour la spécification et la vérification de systèmes distribués soumis à des contraintes temporelles. L'objectif final de ce travail était de permettre une revalorisation de ce modèle, notamment au travers d'une application réelle.

Si la spécification et la validation de protocoles constituent un domaine particulièrement important pour l'utilisation de modèles basés sur les réseaux de Petri, la méthode d'analyse par énumération des classes d'états proposée ici est tout à fait générale et de ce fait peut être appliquée à n'importe quel système dont la modélisation par l'intermédiaire des réseaux de Petri temporels est possible.

En plus des avantages que l'on retrouve dans le formalisme des réseaux de Petri classiques, tels que la représentation aisée du parallélisme et de la concurrence, le large domaine d'applicabilité, la possibilité de modélisation par raffinements successifs, la capacité d'abstraction ou encore la clarté de la représentation graphique, les réseaux de Petri temporels permettent non seulement de représenter des contraintes temporelles mais aussi de simuler le comportement de la plupart des autres modèles.

Quant à la méthode d'analyse, ainsi que nous l'avons déjà dit et répété, elle s'est avérée adéquate dans la plupart des applications traitées jusqu'à ce jour.

Toutefois, à côté de ces qualités, le modèle présente des limites qu'il ne faut pas perdre de vue.

Une première limite est qu'aucune condition nécessaire et suffisante ne peut être établie pour la propriété borné. Or, l'analyse par énumération repose sur cette propriété. Tout au plus, peut-on reculer cette limite en formulant des conditions suffisantes.

Une deuxième limite, typique de l'approche énumérative, c'est que même si l'ensemble des classes d'états est fini, il peut être d'une taille telle qu'il devient difficile à le manipuler et l'exploitation des résultats est alors très délicate.

Ce deuxième inconvénient peut être atténué de diverses manières.

Il est possible de réduire la taille du graphe des classes d'états en apportant une attention toute particulière à la méthodologie de vérification employée [BER 83a]; par exemple, dans le cas de la vérification de systèmes tolérants aux pannes, en traitant séparément chaque cas de panne avec l'hypothèse de panne unique.

D'autres techniques, telles que les méthodes de réduction, qui permettent de réduire la taille d'un réseau tout en conservant certaines de ses propriétés, ont été développées pour les réseaux de Petri classiques et à notre connaissance, sont encore à l'étude en ce qui concerne les réseaux temporels.

De même, il est évident que la manipulation de graphes de classes d'états de taille importante peut être facilitée, ainsi que nous l'avons montré, par l'utilisation d'outils logiciels adéquats.

De tels outils existent mais des améliorations doivent encore y être apportées. Notons que ce n'est que via ces outils que les réseaux de Petri temporels trouveront réellement leur place dans le monde industriel. Un effort particulier doit donc être apporté à leur conception.

Une autre approche pour l'analyse des réseaux de Petri temporels est développée dans [ROU 85b]. Il s'agit de l'approche structurelle qui fait usage de la structure du réseau pour déduire des propriétés sur son comportement sans avoir recours à l'énumération exhaustive des états.

L'avantage de l'approche énumérative sur l'approche structurelle est qu'elle est plus générale et s'applique à tout réseau de Petri temporel borné.

Si ces deux méthodes permettent d'effectuer une analyse qualitative d'un réseau, en revanche, elles sont mal adaptées pour l'analyse de performances. Or, il apparaît que l'on fait de plus en plus appel à ce type d'analyse. Nous ne saurions donc terminer ce travail sans mentionner une extension au modèle que nous avons présenté, et qui contribue à établir le lien manquant entre l'analyse fonctionnelle et l'analyse de performance [ROU 85b].

Cette extension, nommée XTPN (eXtended Time Petri Net), associe à l'intervalle de tir des transitions une fonction densité de probabilité de l'instant de tir. Une méthode d'analyse énumérative permet la construction d'un graphe d'états probabilisé à partir duquel il est possible de prédire les performances du systèmes modélisé [JUA 89].

Nous pensons que la poursuite des recherches dans cette voie devrait assurer un bel avenir aux réseaux de Petri temporels.

BIBLIOGRAPHIE

[ALG 84a]
B. ALGAYRES, J.M. AYACHE, J.P. COURTIAT
Petri Nets are Good for Protocols
LAAS Report n° 3064
ACM SIGCOMM'84 Symposium on Communications Architectures and Protocols, Montreal, June 1984

[ALG 84b]
B. ALGAYRES
Spécification de Protocoles en Couches par Réseaux Prédicats/Transitions
Rapport LAAS n° 84090, Novembre 1984

[AYA 85a]
J.M. AYACHE, P. AZEMA, B. BERTHOMIEU, M. DIAZ, J. DUFAU, B. PRADIN-CHEZALVIEL
Conception et Réalisation d'Outils de Spécification et de Validation des Protocoles
Rapport LAAS n° 85063, Février 1985

[AYA 85b]
J.M. AYACHE, J.P. COURTIAT, M. DIAZ, G. JUANOLE
Utilisation des Réseaux de Petri pour la Modélisation et la Validation des Protocoles
Rapport LAAS n° 85062
Technique et Science Informatiques (numéro Spécial "Réseaux de Petri"), Vol. 4, n° 1, Mars 1985, pp. 51-71

[AYA 86]
J.M. AYACHE, J.P. COURTIAT
Une Méthodologie de Conception des Systèmes Distribués Fondée sur l'Utilisation des Réseaux de Petri
Revue Génie Logiciel, n° 3, Janvier 1986, pp. 36-41

[AZE 88a]
P. AZEMA, J.C. LLORET, F. VERNADAT
Observational and Temporal Logic Verifications
LAAS Report n° 88022, January 1988

[AZE 88b]
P. AZEMA, J.C. LLORET, G. PAPAPANAGIOTAKIS, F. VERNADAT
ESTELLE Validation and PROLOG interpreted Petri Nets
LAAS Report n° 88235, July 1988

[AZE 88c]
P. AZEMA, J.C. LLORET, F. VERNADAT
Réseaux Prédicats Labellés : Structuration et Composition
Rapport LAAS n° 88350, Octobre 1988

[AZE 88d]
P. AZEMA, P. HASTIR, B. JODOCY
La Couche Application de FIP en Réseaux de Petri Labellés à Prédicats
Rapport LAAS n° 88364, Décembre 1988

[AZE 89a]
P. AZEMA, J.C. LLORET, F. VERNADAT
Spécification Logique de Protocoles de Communication à l'Aide de Réseaux Prédicats/Transitions Etiquetés
Rapport LAAS n° 89015, Janvier 1989

[AZE 89b]
P. AZEMA, J.C. LLORET, F. VERNADAT
Spécification par Réseaux Prédicats/Transitions Etiquetés
Rapport LAAS n° 89153, Janvier 1989

[BAR 89]
J.P. BARDINET
FIP : Les Motivations Economiques
Minis & Micros, n° 328, Octobre 1989

[BER 82]
B. BERTHOMIEU, M. MENASCHE
A State Enumeration Approach for Analysing Time Petri Nets
Third European Workshop on Applications and Theory of Petri Nets, Varenna, Italy, September 1982

[BER 83a]
B. BERTHOMIEU, M. MENASCHE
Time Petri Nets for Analysing and Verifying Time Dependent Communication Protocols
Proceedings of the IFIP WG 6.1 Third International Workshop on Protocol Specification, Testing, and Verification, organized by IBM Research, Rüschlikon, Switzerland, 31 May - 2 June, 1983

[BER 83b]
B. BERTHOMIEU, M. MENASCHE
An Enumerative Approach for Analysing Time Petri Nets
IFIP Congress 1983, Paris, North-Holland Ed., September 1983

[BRA 83a]
G.W. BRAMS
Réseau de Petri : Théorie et Pratique, Tome 1 : Théorie et Analyse
Ed. Masson, 1983

[BRA 83b]
G.W. BRAMS
Réseau de Petri : Théorie et Pratique, tome 2 : Modélisation et Applications
Ed. Masson, 1983

[CHR 84]
P. CHRETIENNE
Exécutions Contrôlées des Réseaux de Petri Temporisés
Technique et Science Informatiques, Vol 3, n°1, 1984, pp.23-31

[CIN 89]
F. CINARE
Les Bus de Terrain
Minis & Micros, n° 328, Octobre 1989

[CLO 87]
W. F. CLOCKSIN, C. S. MELLISH
Programming in Prolog
Springer-Verlag, Third Edition, 1987

[COE 88]
H. COELHO, J. C. COTTA
Prolog by Example - How to Learn, Teach and Use it
Springer-Verlag, 1988

[DIA 82]
M. DIAZ
Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models
Computer Networks, Vol. 6, n° 6, December 1982, pp. 419-441

[DIA 84]
M. DIAZ
Petri Net Based Models for the Specification and Validation of Protocols
LAAS Report n° 3161, June 1984

[FAR 76]
D. J. FARBER, P. M. MERLIN
Recoverability of Communication Protocols, Implication of a Theoretical Study
IEEE Transactions on Communications, September 1976, pp. 1036-1043

[FAU 89]
C. FAURE, G. JUANOLE
Modélisation du Transfert de Données de la Couche 2 du RNIS (LAPD)
Rapport LAAS n° 89391, Novembre 1989

[FIC 88]
J. FICHEFET
Introduction aux Réseaux de Petri - Notes de Cours de Première Licence
Facultés Universitaires Notre-Dame de la Paix, Namur, Institut d'Informatique, 1988

[FIP 88a]
CLUB FIP
Spécification des Services Couche Application
Norme n° C 46-602, Dossier Technique, Fevrier 1988

[FIP 88b]
CLUB FIP - LAAS
FIP - Couche Liaison de Données
Modèle en Réseaux de Petri Labellés à Prédicats
Dossier Technique, Octobre 1988

[FIP 89]
CLUB FIP
FIP - Couche Liaison de Données
Norme n° C 46-603, Dossier Technique, Octobre 1989

[GHA 88]
M. GHALLAB, A. MOUNIR ALAOUI
Managing Efficiently Temporal Relations Through Indexed Spanning Trees
LAAS Report n° 88360, December 1988

[HAC 75]
M. HACK
Decision Problems for Petri Nets and Vector Addition Systems
MAC Technical Memoradum 59, MIT, March 1975

[HAS 89]
P. HASTIR, B. JODOCY
Les Réseaux de Petri et leurs Applications
Mémoire de l'Institut d'Informatique des Facultés Universitaires Notre Dame de la Paix, Namur, Juin 1989

[HOA 78]
C.A.R. HOARE
Communicating Sequential Processes
Communications of the ACM, August 1978

[HOL 87]
M. A. HOLLIDAY, M. K. VERNON
A Generalized Timed Petri Net Model for Performance Analysis
IEEE Transactions on Software Engineering, Vol. SE-13, n° 12, December 1987, pp. 1297-1310

[JON 77]
N. D. JONES, L. H. LANDWEBER, Y. E. LIEN
Complexity of Some Problems in Petri Nets
Theoretical Computer Science 4, 1977, pp. 277-299

[JUA 86]
G. JUANOLE, J.L. ROUX
Functional and Performance Analysis Using Extended Time Petri Nets
LAAS Report, n° 86387, December 1886

[JUA 89]
G. JUANOLE, J.L. ROUX
On the Pertinence of the Extended Time Petri Net Model for Analysing Communication Activities
LAAS - VERILOG, 1989

[KAR 69]
R. M. KARP, R. E. MILLER
Parallel Program Schemata
Journal of Computer and System Sciences, Vol. 3, May 1969, pp. 147-195

[LET 89]
P. LETERRIER,J.P. THOMESSE
Services du Système de Communication FIP
Minis & Micros, n° 328, Octobre 1989

[MAR 89]
J.L. MARTHY
Spécification et Analyse d'Architecture de communication pour des Applications Temps Réel
DEA, Automatique Informatique Industrielle et Traitement du Signal, 1989

[MEN 82]
M. MENASCHE
Analyse des Réseaux de Petri Temporisés et Application aux Systèmes Distribués
Thèse de Docteur-Ingénieur, Université Paul Sabatier, Toulouse, Novembre 1985

[MER 74]
P. MERLIN
A Study of the Recoverability of Computer Systems
Ph.D. Thesis, University of California, Irvine, 1974

[MER 75]
P. MERLIN
On the Relation Between Timing and Structure of Recoverable Processes
IBM Thomas Watson Research Center, Yorktown Heights, New York, 1975

[MER 79]
P. MERLIN
Specification and Validation of Protocols
IEEE Transactions on Communications, Vol. COM 27, n° 11, November 1979, pp. 1671-1680

[MUR 89]
T. MURATA
Petri Nets : Properties, Analysis and Applications
Proceedings of the IEEE, Vol. 77, n° 4, April 1989, pp. 541-580

[PAP 86]
G. PAPAPANAGIOTAKIS
Réseaux de Prédicats et Logique Temporelle pour la Vérification des Systèmes Répartis
Thèse de Docteur-Ingénieur, Université Paul Sabatier, Toulouse, Rapport LAAS n° 86334, Novembre 1886

[PET 81]
James L. PETERSON
Petri Net Theory and the Modelling of Systems
Ed. Prentice-Hall, inc., Englewood Cliffs, 1981

[PIP 88]
PIPN - Environnement de Prototypage et de Vérification d'Algorithmes Répartis
LAAS - VERILOG, Manuel de Référence, Version 1, Janvier 1988

[RAM 74]
C. RAMCHANDANI
Analysis of Asynchronous Concurrent Systems by Timed Petri Nets
Ph.D. Thesis, MIT, Project MAC, TR 120, February 1974

[RAZ 83]
R. R. RAZOUK
The Derivation of Performance Expressions for Communication Protocols from Timed Petri Net Models
University of California, Irvine, Technical Report, n° 211, October 1983

[RDP 89]
RdP - Aide à la Conception et l'Evaluation de Systèmes par Réseaux de Petri
VERILOG, Manuel de Référence, Version 1.0, Novembre 1989

[REA 87]
K. REA, R. DE B. JOHNSTON
Automated Analysis of Discrete Communication Behavior
IEEE Transactions on Software Engineering, Vol. SE-13, n° 10, October 1987, pp. 1115-1134

[ROU 85a]
J.L. ROUX
Analyse du Protocole d'Accès au Bus dans le Réseau Local Industriel PROWAY.
Rapport LAAS n° 85002, Janvier 1985

[ROU 85b]
J.L. ROUX
Modélisation et Analyse des Systèmes Distribués par les Réseaux de Petri Temporels
Thèse de Docteur-Ingénieur, Université Paul Sabatier, Toulouse, Rapport LAAS n° 85319, Décembre 1985

[SHA 86]
E. SHAPIRO, L. STERLING
The Art of Prolog : Advanced Programming Techniques
MIT Press, Cambridge, Massachusetts, 1986

[SIF 77]
J. SIFAKIS
Use of Petri Nets for Performance Evaluation in Measuring Modelling and Evaluating Computer Systems
North-Holland Publ. Co., 1977, pp. 75-93

[SIF 79]
J. SIFAKIS
Le Contrôle des Systèmes Asynchrones : Concepts, Propriétés, Analyse Statique
Thèse d'Etat, Université Scientifique et Médicale, Grenoble, Juin 1979

[VAL 79]
R. VALETTE
Analysis of Petri Nets by Stepwise Refinements
Journal of Computer and System Sciences, Vol. 18, n° 1, February 1979, pp. 35-46

[VER 89]
F. VERNADAT
Vérification Formelle d'Applications Réparties - Caractérisation Logique d'une Equivalence de Comportement
Thèse de Doctorat, Université Paul Sabatier, Toulouse, rapport LAAS n° 89386, Novembre 1989

[ZIM 80]
H. ZIMMERMAN
OSI Reference Model. The ISO Model of Architecture for Open Systems Interconnection
IEEE Transactions on Communications, Com-28, April 1980

[ZUB 80]
W. M. ZUBEREK
Timed Petri Nets and Preliminary Performance Evaluation
Seventh Annual Symposium on Computer Architecture, La Baule, France, May 1980, pp. 88-96

ANNEXES

ANNEXE 1

TEXTE SOURCE DU SIMULATEUR

TEXTE SOURCE DU SIMULATEUR

Notations particulières

- le triplet (_Ai,_Ti,_Bi) représente la relation $a_i \leq t_i \leq b_i$ où a_i est la date de tir au plus tôt et b_i la date de tir au plus tard de la transition t_i.
- le triplet (_Ti,_Tj,_Cij) représente la relation d'interdépendance $t_i - t_j \leq c_{ij}$ entre les transitions t_i et t_j.
- _LT est la liste de toutes les transitions du réseau de Petri temporel.
- _LTS est la liste des transitions sensibilisées.
- _M est le marquage du réseau.
- _DT est la liste des inégalités de transitions.
- _DI est la liste des relations d'interdépendance entre les transitions.
- _D est le domaine de tir associé à la classe d'état considérée (_D = _DT ∪ _DI).
- '*' est le symbole utilisé pour représenter un nombre infini.

```
{ P1 : max(_X,_Y,_Y) :-
            _Y est le maximum de _X et de _Y.                    }

    max('*',_Y,'*') :- !.
    max(_X,'*','*') :- !.
    max(_X,_Y,_X) :-
            _X≥_Y,!.
    max(_X,_Y,_Y).
```

```
{ P2 : min(_X,_Y,_Y) :-
            _Y est le minimum de _X et de _Y.                    }

    min('*',_Y,_Y) :- !.
```

```
min(_X,'*',_X) :- !.
min(_X,_Y,_X) :-
        _X ≤ _Y,!.
min(_X,_Y,_Y).
```

{ P3 : moins(_A,_B,_C) :-
_C est le résultat de _A-_B. }

```
moins(_A,_B,_C) :-
        number(_A),
        number(_B),!,
        _C is _A-_B.
moins('*',_B,'*') :-
        number(_B),!.
moins(_A,'*',-'*') :-
        number(_A),!.
```

{ P4 : plus(_A,_B,_C) :-
_C est le résultat de _A+_B. }

```
plus(_A,_B,_C) :-
        number(_A),
        number(_B),!,
        _C is _A+_B.
plus('*',_B,'*') :-
        number(_B),!.
plus(_A,'*','*') :-
        number(_A),!.
```

{ P5 : égal(_X,_X) :-
les deux termes sont égaux. }

```
égal(_X,_X).
```

{ P6 : nb(_X,_L,_N) :-
_N est le nombre d'éléments _X dans _L. }

```
nb(_X,[ ],0) :- !.
```

```
nb(_X,[_X|_Q],_N) :-
        nb(_X,_Q,_N1),
        _N is _N1+1,!.
nb(_X,[_Y|_Q],_N) :-
        nb(_X,_Q,_N).
```

{ P7 : nb_occ(_L1,_L2,_LSD1) :-

le nombre d'occurrences des éléments de _L2 est ≥ au nombre d'occurrences des éléments de _L1.

On utilise la liste _LSD1, c'est-à-dire la liste _L1 de laquelle on a retiré les doubles. **}**

```
nb_occ(_L1,_L2,[ ]) :- !.
nb_occ(_L1,_L2,[_T|_Q]) :-
        nb(_T,_L1,_N1),
        nb(_T,_L2,_N2),
        _N2≥_N1,
        nb_occ(_L1,_L2,_Q).
```

{ P8 : membre(_X,_L) :-

_X est dans _L. **}**

```
membre(_X,[_X|_]).
membre(_X,[_|_Q]) :-
        membre(_X,_Q).
```

{ P9 : inclusion_l(_L1,_L2) :-

_L1 est inclue dans _L2. **}**

```
inclusion_l([ ],_L) :- !.
inclusion_l([_X|_Q1],_L2) :-
        membre(_X,_L2),
        inclusion_l(_Q1,_L2).
```

{ P10 : inclusion_l_d(_L1,_L2,_LSD1,_LSD2) :-

_L1 est inclue dans _L2 et les deux listes contiennent des éléments doubles. _LSD1 et _LSD2 sont les listes _L1, _L2 dans lesquelles on a supprimé les éléments doubles. On vérifie ainsi l'inclusion de _LRES1 dans _LRES2 et

le prédicat nb_occ permet de tester que le nombre d'occurrences des éléments de _L2 est ≥ au nombre d'occurrences des éléments de _L1. }

```
inclusion_l_d(_L1,_L2,_LSD1,_LSD2) :-
        supprimer_doubles(_L1,_LSD1),
        supprimer_doubles(_L2,_LSD2),
        inclusion_l(_LSD1,_LSD2),
        nb_occ(_L1,_L2,_LSD1).
```

{ P11 : supprimer_élt(_X,_L,_LRES) :-

_LRES est la liste _L dans laquelle on a supprimé la première occurrence de l'élément _X. }

```
supprimer_élt(_X,[_X],[ ]) :- !.
supprimer_élt(_X,[_X | _Q],_Q) :- !.
supprimer_élt(_X,[_Y | _Q],[_Y | _Q1]) :-
        supprimer_élt(_X,_Q,_Q1).
```

{ P12 : supprimer_doubles(_L,_LSD) :-

_LSD est la liste _L dans laquelle on a supprimé tous les doubles.

}

```
supprimer_doubles([ ],[ ]) :- !.
supprimer_doubles([_X],[_X]) :- !.
supprimer_doubles([_X | _Q],_L) :-
        membre(_X,_Q),!,supprimer_doubles(_Q,_L).
supprimer_doubles([_X | _Q],[_X | _Q1]) :-
        supprimer_doubles(_Q,_Q1).
```

{ P13 : soustraire_l(_L1,_L2,_L) :-

_L est la liste obtenue en soustrayant _L2 de _L1.

Pour pouvoir faire la différence, il faut que _L2 soit incluse dans _L1. Le prédicat inclusion_l_d(_L2,_L1,_LSD1,_LSD2) permet de tester l'inclusion de _L2 dans _L1. }

```
soustraire_l(_L,[ ],_L) :- !.
soustraire_l(_L1,_L2,_L) :-
        inclusion_l_d(_L2,_L1,_LSD1,_LSD2),!,
        soustraire_l_2(_L1,_L2,_L).
```

```
soustraire_l(_L1,_L2,_L) :-
        write('Erreur !'),
        fail.
```

{ P14 : soustraire_l_2(_L1,_L2,_L) :-
_L est la liste obtenue en soustrayant _L2 de _L1.
Les listes _L1 et _L2 sont telles que _L2 est incluse dans _L1.
Si un élément _T1 n'appartient pas à _L2, il appartient à la différence; s'il appartient à _L2, il n'appartient pas à la différence. }

```
soustraire_l_2(_L,[ ],_L) :- !.
soustraire_l_2([_T1 | _Q1],_L2,[_T1 | _Q]) :-
        not membre(_T1,_L2),!,
        soustraire_l_2(_Q1,_L2,_Q).
soustraire_l_2([_T1 | _Q1],_L2,_L) :-
        supprimer_élt(_T1,_L2,_LRES),
        soustraire_l_2(_Q1,_LRES,_L).
```

{ P15 : intersection(_L1,_L2,_L) :-
_L est l'intersection de _L1 et _L2. }

```
intersection(_L,[ ],[ ]) :- !.
intersection([ ],_L,[ ]) :- !.
intersection([_T1 | _Q1],_L2,[_T1 | _Q] :-
        membre(_T1,_L2),!,
        supprimer_élt(_T1,_L2,_LR2),
        intersection(_Q1,_LR2,_Q).
intersection([_T1 | _Q1],_L2,_L) :-
        intersection(_Q1,_L2,_L).
```

{ P16 : concat(_L1,_L2,_L) :-
_L est la concaténation de _L1 et _L2. }

```
concat([_X | _L1],_L2,[_X | _L3]) :-
        concat(_L1,_L2,_L3).
concat([ ],_L,_L).
```

{ P17 : trans(_Ti) :-

la transition _Ti est déclarée dans le corps du module décrivant le réseau de Petri temporel. }

```
trans(_TI) :-
        tr(_Ti,_,_,_).
```

{ P18 : tr_delay(_Ai,_Ti,_Bi) :-

_Ai et _Bi représentent les contraintes temporelles associées à la transition _Ti qui est déclarée dans le corps du module décrivant le réseau de Petri. }

```
tr_delay(_Ai,_Ti,_Bi) :-
        tr(_Ti,_,_,delay(_Ai,_Bi)).
```

{ P19 : tr_pré(_Ti,_PRE) :-

_PRE est la précondition de la transition _Ti qui est déclarée dans le corps du module décrivant le réseau de Petri temporel. }

```
tr_pré(_Ti,_PRE) :-
        tr(_Ti,pré(_PRE),_,_).
```

{ P20 : tr_pré_post(_Ti,_PRE,_POST) :-

_PRE et _POST sont respectivement les précondition et postcondition de la transition _Ti qui est déclarée dans le corps du module décrivant le réseau de Petri temporel. }

```
tr_pré_post(_Ti,_PRE,_POST) :-
        tr(_Ti,pré(_PRE),post(_POST),_).
```

{ P21 : transitions(_LT) :-

_LT est la liste de toutes les transitions déclarées dans le corps du module décrivant le réseau de Petri. C'est la liste de toutes les instances _Ti pour lesquelles le prédicat trans(_Ti) est vérifié. }

```
transitions(_LT) :-
        setof(_Ti,trans(_Ti),_LT).
```

{ P22 : intervalle((_Ai,_Ti,_Bi),_LT) :-

_Ai et _Bi sont respectivement les bornes inférieure et supérieure de l'intervalle associé la transition _Ti qui appartient à la liste _LT. }

```
intervalle((_Ai,_Ti,_Bi),_LT) :-
        tr_delay(_Ai,_Ti,_Bi),
        membre(_Ti,_LT).
```

{ P23 : transition_sensibilisée(_Ti,_M) :-

_Ti est sensibilisée dans le marquage _M; le marquage vérifie la précondition de la transition. }

```
transition_sensibilisée(_Ti,_M) :-
        tr_pré(_Ti,_PRE),
        inclusion_l_d(_PRE,_M,_PRESD,_MSD).
```

{ P24 : l_transitions_sensibilisées(_LTS,_M) :-

_LTS est la liste des transitions sensibilisées dans le marquage _M. }

```
l_transitions_sensibilisées(_LTS,_M) :-
        setof(_Ti,transition_sensibilisée(_Ti,_M),_LTS),!.
l_transitions_sensibilisées([ ],_M).
```

{ P25 : domaine(_D,_LTS) :-

_D est le domaine de tir (liste dont les éléments sont de la forme (_Ai,_Ti,_Bi)) défini pour les transitions sensibilisées qui appartiennent à la liste _LTS. }

```
domaine(_D,_LTS) :-
        setof((_Ai,_Ti,_Bi),intervalle((_Ai,_Ti,_Bi),_LTS),_D),!.
domaine([ ],_LTS).
```

{ P26 : cohérent(_L) :-

la liste _L des éléments de la forme (_Ai,_Ti,_Bi) est cohérente; c'est-à-dire que pour chaque élément on a : _Ai ≥ 0 et _Ai ≤ _Bi. }

```
cohérent([ ]) :- !.
```

```
cohérent((_Ai,_Ti,'*')) :-
        _Ai≥0,!.
cohérent((_Ai,_Ti,_Bi)) :-
        _Ai≥0,
        _Bi≥_Ai,!.
cohérent([_T|_Q]) :-
        cohérent(_T),
        cohérent(_Q).
```

{ P27 : éliminer_l_tr(_L,_DI,_DT,_DIN,_DTN) :-
_DIN et _DTN sont les listes obtenues à partir de _DI et _DT en y éliminant les éléments contenus dans la liste _L. }

```
éliminer_l_tr([ ],_DI,_DT,_DI,_DT) :- !.
éliminer_l_tr(_L,_DI,[ ],_DI,[ ]) :- !.
éliminer_l_tr(_L,[ ],_DT,[ ],_DT) :- !.
éliminer_l_tr([_Te|_Q],_DI,_DT,_DIN,_DTN) :-
        membre((_Ae,_Te,_Be),_DT),
        éliminer_tr_DT((_Ae,_Te,_Be),_DI,_DT,_DTN1),
        éliminer_tr_DI((_Te,_DI,_DIN1),
        éliminer_l_tr(_Q,_DIN1,_DTN1,_DIN,_DTN).
```

{ P28 : membre_DT((_Ai,_Ti,_Bi),_DT,_L) :-
_Ti est un élément de _L et (_Ai,_Ti,_Bi) un élément de _DT. }

```
membre_DT((_Ai,_Ti,_Bi),_DT,_L) :-
        membre(_Ti,_L),
        membre((_Ai,_Ti,_Bi),_DT).
```

{ P29 : réduire_DT(_DT,_L,_DTN) :-
_DTN est la réduction du domaine _DT; seules les transitions appartenant à _L y sont représentées. }

```
réduire_DT(_DT,_L,_DTN) :-
        setof((_Ai,_Ti,_Bi),membre_DT((_Ai,_Ti,_Bi),_DT,_L),_DTN),!.
réduire_DT(_DT,_L,[ ]).
```

{ P30 : modifier_DT((_Ai,_Ti,_Bi),_DT,_DTN,_DI) :-

_DTN est le nouveau domaine de tir obtenu à partir de l'ancien domaine _DT en y appliquant l'étape 2.2 de l'algorithme du calcul de la classe suivante (relations d'interdépendance non comprises). }

```
modifier_DT((_Ai,_Ti,_Bi),[ ],[ ],_DI) :- !.
modifier_DT((_Ai,_Ti,_Bi),[(_Ai,_Ti,_Bi) | _Q],_DTN,_DI) :-
        modifier_DT((_Ai,_Ti,_Bi),_Q,_DTN,_DI),!.
modifier_DT((_Ai,_Ti,_Bi),[(_Aj,_Tj,_Bj)],[(_Ajn,_Tj,_Bjn)],_DI) :-
        moins(_Aj,_Bi,_BUF1),
        max(0,_BUF1,_BUF2),
        membre((_Ti-_Tj,_Cij),_DI),
        max(_BUF2,-_Cij,_Ajn),
        membre((_Tj-_Ti,_Cji),_DI),
        moins(_Bj,_Ai,_BUF3),
        min(_Cji,_BUF3,_Bjn),!.
modifier_DT((_Ai,_Ti,_Bi),[_T | _Q],[_TN | _QN],_DI) :-
        modifier_DT((_Ai,_Ti,_Bi),[_T],[_TN],_DI),
        modifier_DT((_Ai,_Ti,_Bi),_Q,_QN,_DI).
```

{ P31 : éliminer_tr_DT((_Ae,_Te,_Be),_DI,_DT_DTN) :-

_DI reste inchangé.

_DTN est obtenue en appliquant à _DT les changements nécessaires pour l'élimination d'une transition du domaine de tir. Ces changements correspondent à l'étape 2.3. de l'algorithme du calcul de la classe suivante (relations d'interdépendance non comprises).

}

```
éliminer_tr_DT((_Ae,_Te,_Be),_DI,[ ],[ ]) :- !.
éliminer_tr_DT((_Ae,_Te,_Be),[ ],_DT,_DT) :- !.
éliminer_tr_DT((_Ae,_Te,_Be),_DI,[(_Ae,_Te,_Be) | _Q],[(_Ae,_Te,_Be) | _QN]) :-
        éliminer_tr_DT((_Ae,_Te,_Be),_DI,_Q,_QN),!.
éliminer_tr_DT((_Ae,_Te,_Be),_DI,[(_Aj,_Tj,_Bj) | _Q],[(_Ajn,_Tj,_Bjn) | _QN] :-
        membre((_Te-_Tj,_Cej),_DI),
        moins(_Ae,_Cej,_BUF1),
        max(_Aj,_BUF1,_Ajn),
        membre((_Tj-_Te,_Cje),_DI),
        plus(_Be,_Cje,_BUF2),
        min(_Bj,_BUF2,_Bjn),
```

```
éliminer_tr_DT((_Ae,_Te,_Be),_DI,_Q,_QN).
```

{ P32 : réd_DI(_Te,_DI,_DIN) :-

_DIN est la liste obtenue à partir de _DI en y éliminant les relations d'interdépendance relatives à la transition _Te. }

```
réd_DI(_Te,[ ],[ ]) :- !.
réd_DI(_Te,[(_Te-_Tj,_Cej) | _Q],_DIN) :-
        réd_DI(_Te,_Q,_DIN),!.
réd_DI(_Te,[(_Tj-_Te,_Cje) | _Q],_DIN) :-
        réd_DI(_Te,_Q,_DIN),!.
réd_DI(_Te,[(_Tk-_Tl,_Ckl) | _Q],[_Tk-_Tl,_Ckl) | _QN]) :-
        réd_DI(_Te,_Q,_QN).
```

{ P33 : réduire_DI(_L,_DI,_DIN) :-

_DIN est la liste obtenue à partir de _DI en y éliminant les relations d'interdépendance relatives aux transitions de la liste _L. }

```
réduire_DI([ ],_DI,_DI) :- !.
réduire_DI([_Te | _Q],_DI,_DIN) :-
        réd_DI(_Te,_DI,_DIN1),réduire_DI(_Q,_DIN1,_DIN).
```

{ P34 : modifier_DI((_Ai,_Ti,_Bi),_DT,_DI,_DIN) :-

_DT reste inchangée.

_DIN est la liste des relations d'interdépendance du nouveau domaine de tir, obtenue à partir de _DI en y appliquant l'étape 2.2 de l'algorithme du calcul de la classe suivante; on traite ici uniquement les relations d'interdépendance. }

```
modifier_DI(_,_,[ ],[ ]) :- !.
modifier_DI((_Ai,_Ti,_Bi),_DT,[_Ti-_Tk,_Cik) | _Q],_DIN) :-
        modifier_DI((_Ai,_Ti,_Bi),_DT,_Q,_DIN),!.
modifier_DI((_Ai,_Ti,_Bi),_DT,[(_Tk-_Ti,_Cki) | _Q],_DIN) :-
        modifier_DI((_Ai,_Ti,_Bi),_DT,_Q,_DIN),!.
modifier_DI((_Ai,_Ti,_Bi),_DT,[(_Tk-_Tl,_Ckl)],[(_Tk-_Tl,_Ckln)] :-
        membre((_Ak,_Tk,_Bk),_DT),
        membre((_Al,_Tl,_Bl),_DT),
        moins(_Bk,_Al,_BUF),
```

```
            min(_Ckl,_BUF,_Ckln),!.
modifier_DI((_Ai,_Ti,_Bi),_DT,[_T|_Q],[_TN|_QN]) :-
            modifier_DI((_Ai,_Ti,_Bi),_DT,[_T],[_TN],
            modifier_DI((_Ai,_Ti,_Bi),_DT,_Q,_QN).
```

{ P35 : éliminer_tr_DI(_Te,_DI,_DIN) :-

_DIN est la liste obtenue à partir de _DI en y appliquant les changements nécessaires pour l'élimination d'une transition. Ces changements correspondent au point 2.3. de l'algorithme du calcul de la classe suivante; seules les relations d'interdépendance sont prises en compte.

}

```
éliminer_tr_DI(_Te,[ ],[ ]) :- !.
éliminer_tr_DI(_Te,[(_Te-_Ti,_Cei)|_Q],[(_Te-_Ti,_Cei)|_QN]) :-
            éliminer_tr_DI(_Te,_Q,_QN),!.
éliminer_tr_DI(_Te,[(_Ti-_Te,_Cie)|_Q],[(_Ti-_Te,_Cie)|_QN]) :-
            éliminer_tr_DI(_Te,_Q,_QN),!.
éliminer_tr_DI(_Te,[(_Tk-_Tl,_Ckl)|_Q],[(_Tk-_Tl,Ckln)|_QN]) :-
            membre((_Tk-_Te,_Cke),_Q),
            membre((_Te-_Tl,_Cel),_Q),
            plus(_Cke,_Cel,_BUF),
            min(_Ckl,_BUF,_Ckln),
            éliminer_tr_DI(_Te,_Q,_QN).
```

{ P36 : nouveau_DI(_L,_DT,_DIN) :-

_DIN est la liste des nouvelles relations d'interdépendance entre les transitions de la liste _L et les transitions de la liste _DT. **}**

```
nouveau_DI([ ],_,[ ]) :- !.
nouveau_DI(_,[ ],[ ]) :- !.
nouveau_DI([(_Ai,_Ti,_Bi)],[(_Ai,_Ti,_Bi)|_Q],_DIN) :-
            nouveau_DI([(_Ai,_Ti,_Bi)],_Q,_DIN),!.
nouveau_DI([(_Ak,_Tk,_Bk)],[(_Al,_Tl,_Bl)],[(_Tk-_Tl,_Ckl)]) :-
            moins(_Bk,_Al,_Ckl),!.
nouveau_DI([(_Ak,_Tk,_Bk)],[_T|_Q],_DIN) :-
            nouveau_DI([(_Ak,_Tk,_Bk)],[_T],_DIN1),
            nouveau_DI([(_Ak,_Tk,_Bk)],_Q,_DIN2),
            concat(_DIN1,_DIN2,_DIN),!.
```

```
nouveau_DI([_T | _Q],_DT,_DIN) :-
        nouveau_DI([_T],_DT,_DIN1),
        nouveau_DI(_Q,_DT,_DIN2),
        concat(_DIN1,_DIN2,_DIN).
```

{ P37 : interdép(_D,(_Ai,_Ti,_Bi),_RI) :-
_RI est la liste des relations d'interdépendance entre la transition _Ti et celles contenues dans _D. }

```
interdép([ ],_,[ ]) :- !.
interdép([(_Ai,_Ti,_Bi) | _Q],(_Ai,_Ti,_Bi),_RI) :-
        interdép(_Q,(_Ai,_Ti,_Bi),_RI),!.
interdép([_Aj,_Tj,_Bj) | _Q],(_Ai,_Ti,_Bi),[(_Ti-_Tj,_Cij) | _QI]) :-
        moins(_Bi,_Aj,_Cij),
        interdép(_Q,(_Ai,_Ti,_Bi),_QI),!.
```

{ P38 : l_interdép(_D1,_D2,_RI) :-
_RI est la liste des relations d'interdépendance entre les transitions de _D1 et celles de _D2. }

```
l_interdép([ ],_D,[ ]) :- !.
l_interdép([_T | _Q],_D,_RI) :-
        interdép(_D,_T,_RI1),
        l_interdép(_Q,_D,_RI2),
        concat(_RI1,RI2,_RI).
```

{ P39 : choix(_Ti,_LTS) :
cette règle affiche à l'écran la liste _LTS des transitions sensibilisées et invite l'utilisateur à faire son choix. }

```
choix(_Ti,_LTS) :-
        write('Les transitions sensibilisées sont : '),
        nl,
        write(_LTS),
        nl,
        write('Choisissez une transition ou tapez "stop" pour finir :'),
        erreur(_Ti,_LTS).
```

{ P40 : stop(_Ti,_LTS) :

cette règle arrête l'exécution du simulateur si _Ti est égal à stop. }

```
stop(_Ti,_LTS) :-
        égal(_Ti,stop),!,
        halt.
stop(_Ti,_LTS).
```

{ P41 : erreur(_Ti,_LTS) :

si _Ti est différent de stop et n'est pas une transition sensibilisée, la liste _LTS des transitions sensibilisées est réaffichée et l'utilisateur est invité à formuler un nouveau choix. }

```
erreur(_Ti,_LTS) :-
        read(_Ti),
        stop(_Ti,_LTS),
        membre(_Ti,_LTS),!.
erreur(_Ti,_LTS) :-
        nl,
        write('Erreur ! Cette transition n''est pas sensibilisée.'),
        nl,
        write(_LTS),
        nl,
        write('Nouveau choix : '),
        erreur(_Ti,_LTS).
```

{ P42 : classe_suivante(_M,_LT,_LTS,_Ti,_DT,_DI,_MS,_DTS,_DIS) :

cette règle calcule la nouvelle classe d'états engendrée par le tir de la transition _Ti. }

```
classe_suivante(_M,_LT,_LTS,_Ti,_DT,_DI,_MS,_DTS,_DIS) :-
        tr_pre_post(_Ti,_PRE,_POST),
        membre((_Ai,_Ti,_Bi),_DT),
        modifier_DT((_Ai,_Ti,_Bi),_DT,_DT1,_DI),
        modifier_DI(_(_Ai,_Ti,_Bi),_DT1,_DI,_DI1),
        soustraire_l(_M,_PRE,_M1),
        l_transitions_sensibilisées(_LTS1,_M1),
        {_LTS1 est la liste des transitions restées sensibilisées}
        soustraire_l(_LTS,_LTS1,_BUF),
```

```
        soustraire_l(_BUF,[_Ti],_LTD1),
        {_LTD1 est la liste des transitions désensibilisées par le tir de _Ti}
        éliminer_l_tr(_LTD1,_DI1,_DT1,_DI2,_DT2),
        soustraire_l(_LTS,_BUF,_LTS2),
        réduire_DT(_DT2,_LTS2,_DT3),
        réduire_DI(_LTD1,_DI2,_DI3),
        concat(_M1,_POST,_MS),
        l_transitions_sensibilisées(_LTS3,_MS),
        soustraire_l(_LTS3,_LTS1,_LTS4),
        {_LTS4 est la liste des transitions nouvellement sensibilisées}
        domaine(_DT4,_LTS4),
        concat(_DT3,_DT4,_DTS),
        nouveau_DI(_DT4,_DTS,_DI4),
        soustraire_l(_DTS,_DT4,_BUF2),
        nouveau_DI(_BUF2,_DT4,_DI5),
        concat(_DI4,_DI5,_DI6),
        concat(_DI3,_DI6,_DIS).
```

{ P43 : simulation(_M,_LT,_DT,_DI,_MS,_DTS,_DIS) :

cette règle affiche la liste des transitions sensibilisées et demande à l'utilisateur d'en choisir une. Quand c'est chose faite, elle appelle la procédure classe_suivante qui calcule la classe d'états suivante en vérifiant que celle-ci est cohérente. }

```
simulation(_M,_LT,_DT,_DI,_MS,_DTS,_DIS) :-
        l_transitions_sensibilisées(_LTS,_M),
        choix(_Ti,_LTS),
        classe_suivante(_M,_LT,_LTS,_Ti,_DT,_DI,_MS,_DTS,_DIS),
        write('Classe suivante : '),
        nl,
        write('M : '),
        write(MS),
        nl,
        concat(_DTS,_DIS,_D),
        write('D =  '),
        write(_D),
        nl,
        cohérent(_DTS),
        !,
```

```
        nl,nl,
        simulation(_MS,_LT,_DTS,_DIS,_MS2,_DTS2,_DIS2).
simulation(_M,_LT,_DT,_DI,_MS,_DTS,_DIS) :-
        write('Le domaine calculé est incohérent !'),
        nl,
        write('Le simulateur considère la classe précédente'),
        nl,
        write('M : '),
        write(_M),
        nl,
        write('D : '),
        concat(_DT,_DI,_D),
        write(_D),
        nl,nl,
        simulation(_M,_LT,_DT,_DI,_MS2,_DTS2,_DIS2).
```

{ P44 : simu :

une fois que l'utilisateur a indiqué le nom du fichier qui contient la description du réseau de Petri temporel, la simulation et le calcul des classes d'états de ce réseau peuvent commencer. **}**

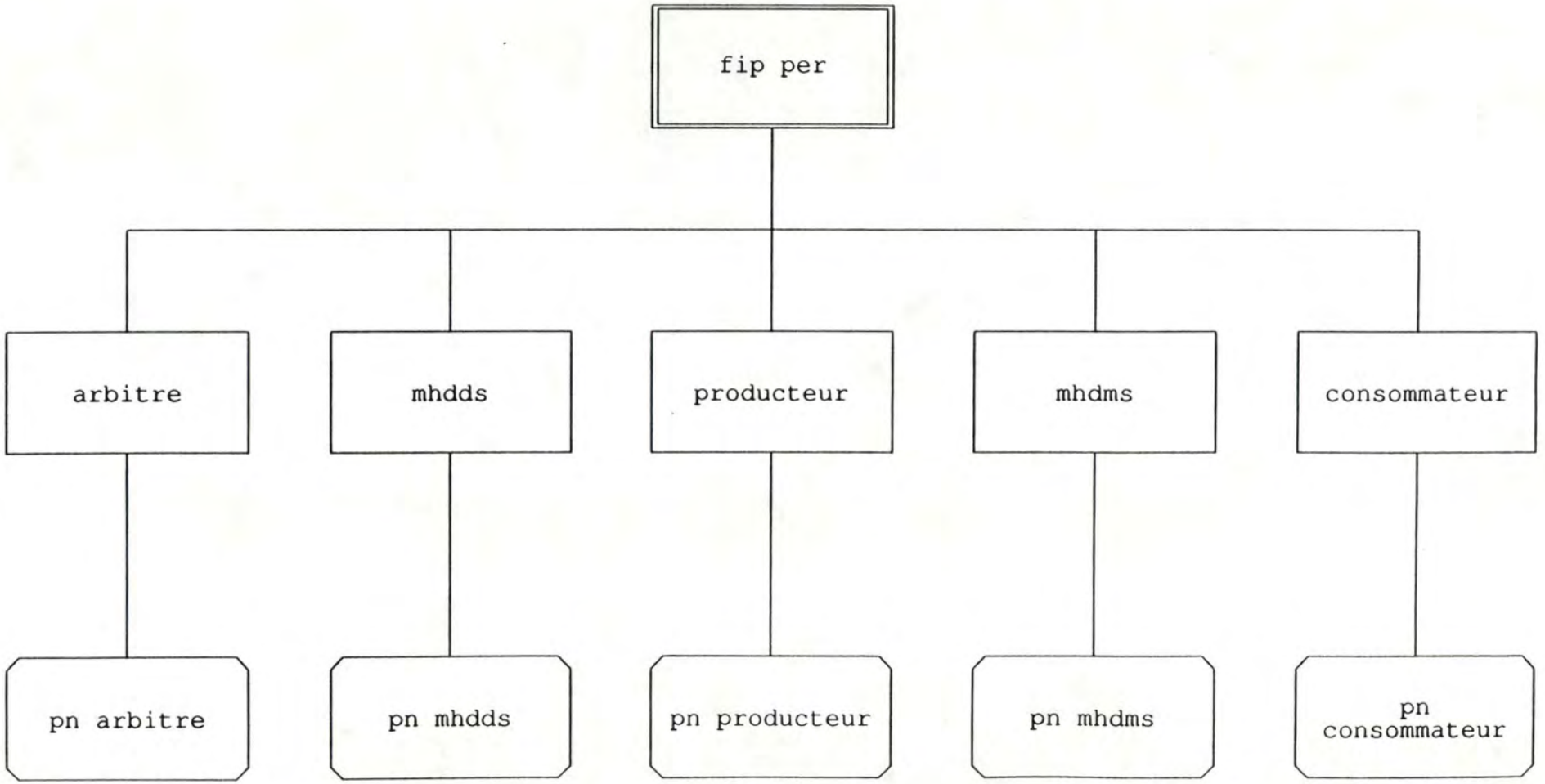
```
simu :-
        write('Entrez le nom du fichier contenant la description du réseau de Petri
        temporel : '),
        nl,
        read(_F),
        consult(_F),
        transitions(_LT),
        init(_M),
        l_transitions_sensibilisées(_LTS,_M),
        domaine(_DT,_LTS),
        l-interdép(_DT,_DT,_DI),
        concat(_DT,_DI,_D),
        write('La classe initiale est : '),
        nl,
        write('M0 = '),
        write(_M),
        nl,
        write('D0 = '),
```

```
write(_D),
nl,nl,nl,
simulation(_M,_LT,_DT,_DI,_MS,_DTS,_DIS).
```

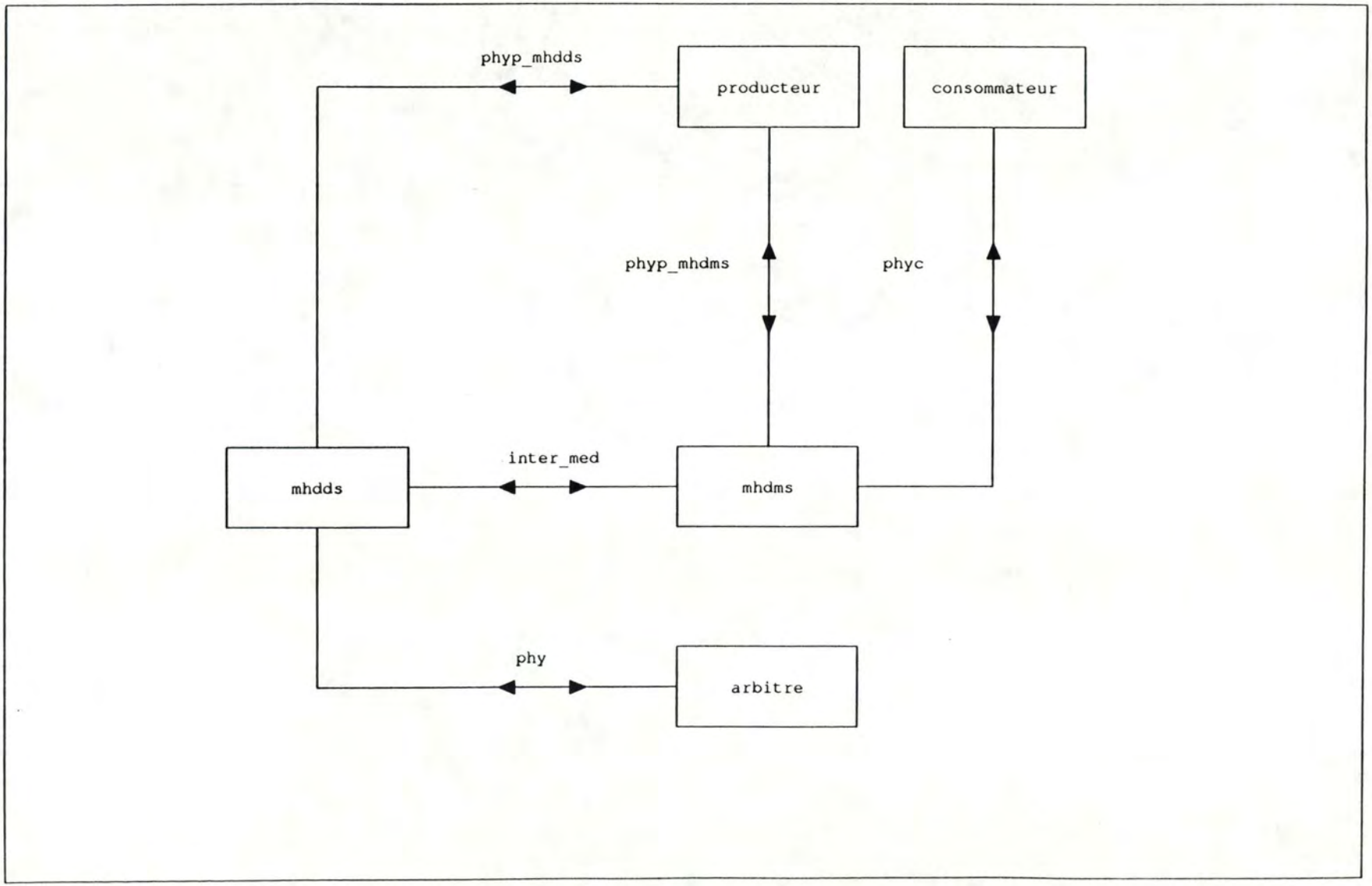
ANNEXE 2

MODELE FIP_PER - ARCHITECTURE

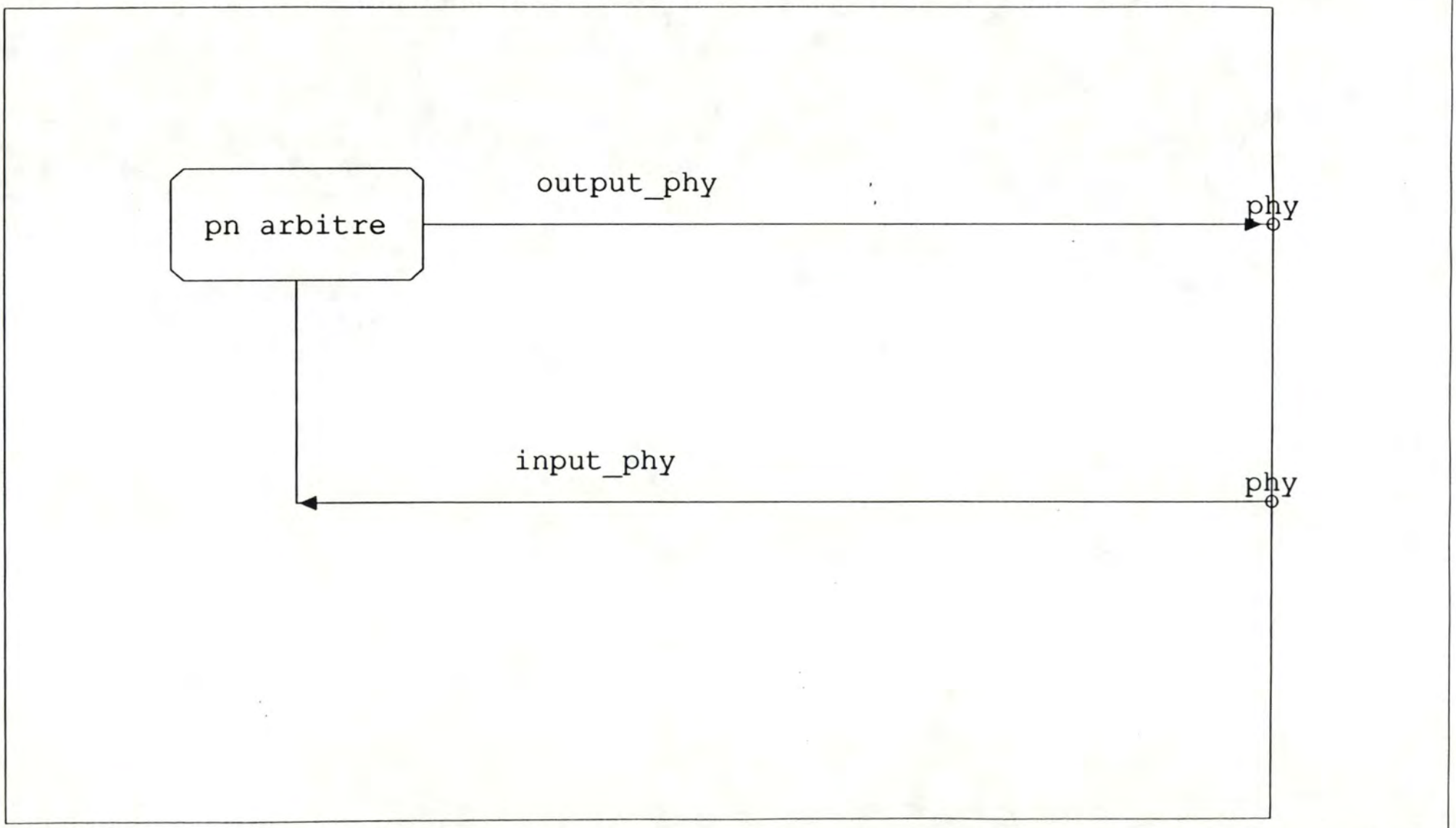
	DESCRIPTION: fip_per TYPE: time	Page: 1
1.0	SYSTEM fip_per	26-Dec-1989



	DESCRIPTION: fip_per TYPE: time	Page: 2
1.0	SYSTEM fip_per	26-Dec-1989



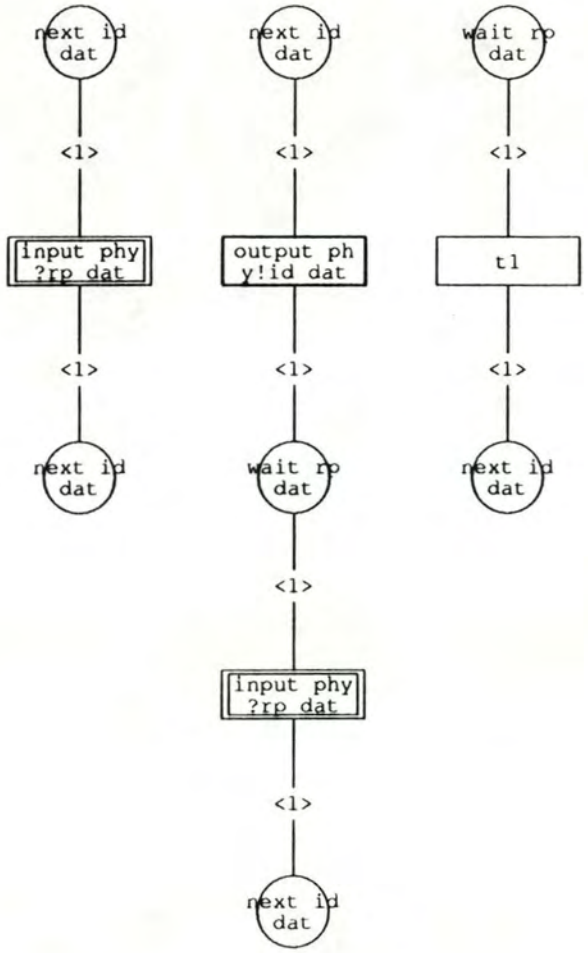
	DESCRIPTION: fip_per TYPE: time	Page: 3
1.0	BLOCK fip_per/arbitre	26-Dec-1989



	DESCRIPTION: fip_per TYPE: time	Page: 4
1.0	PETRI NET fip_per/arbitre/pn_arbitre	26-Dec-1989

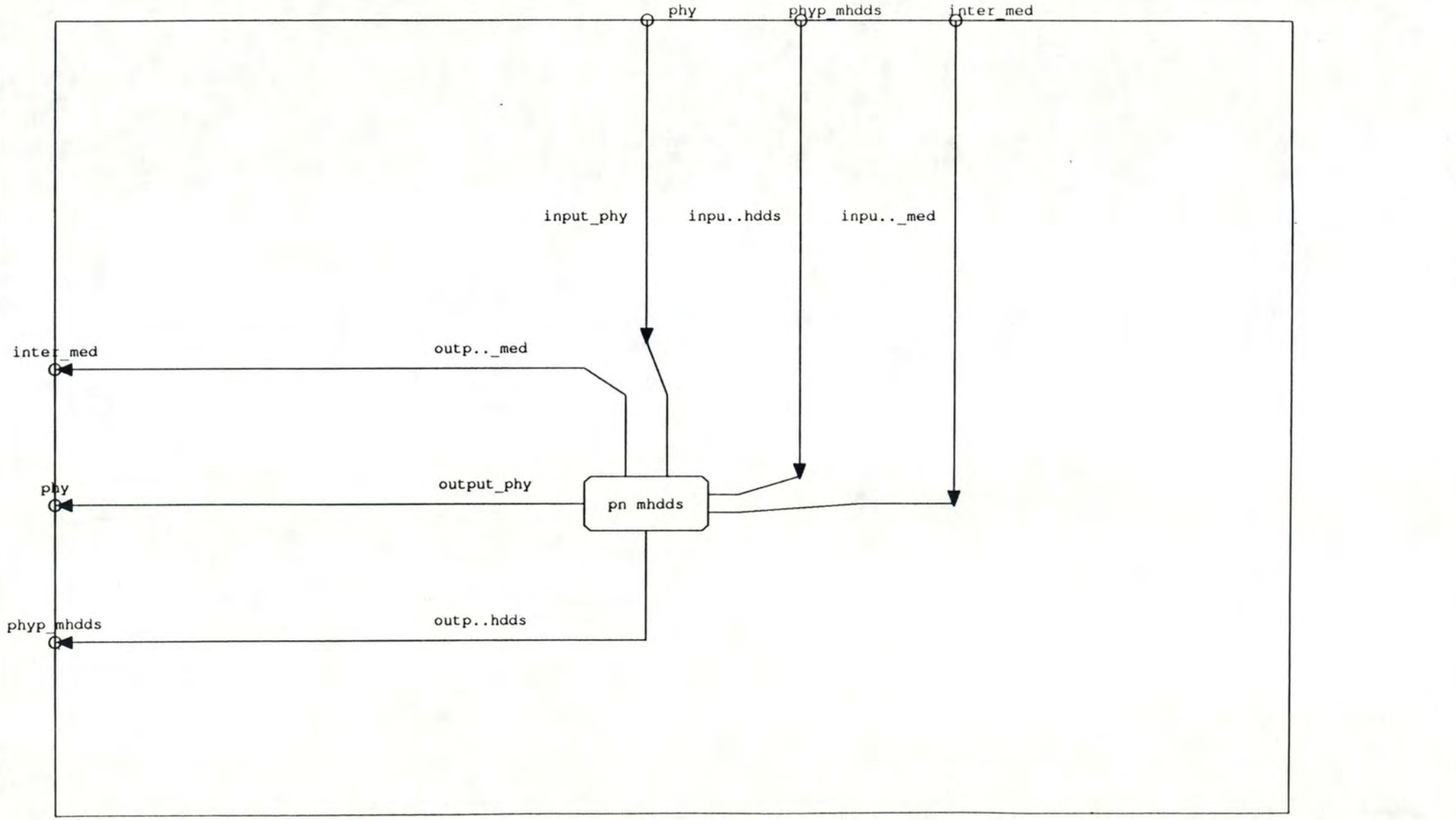
⊕ out..phy

inp..dat

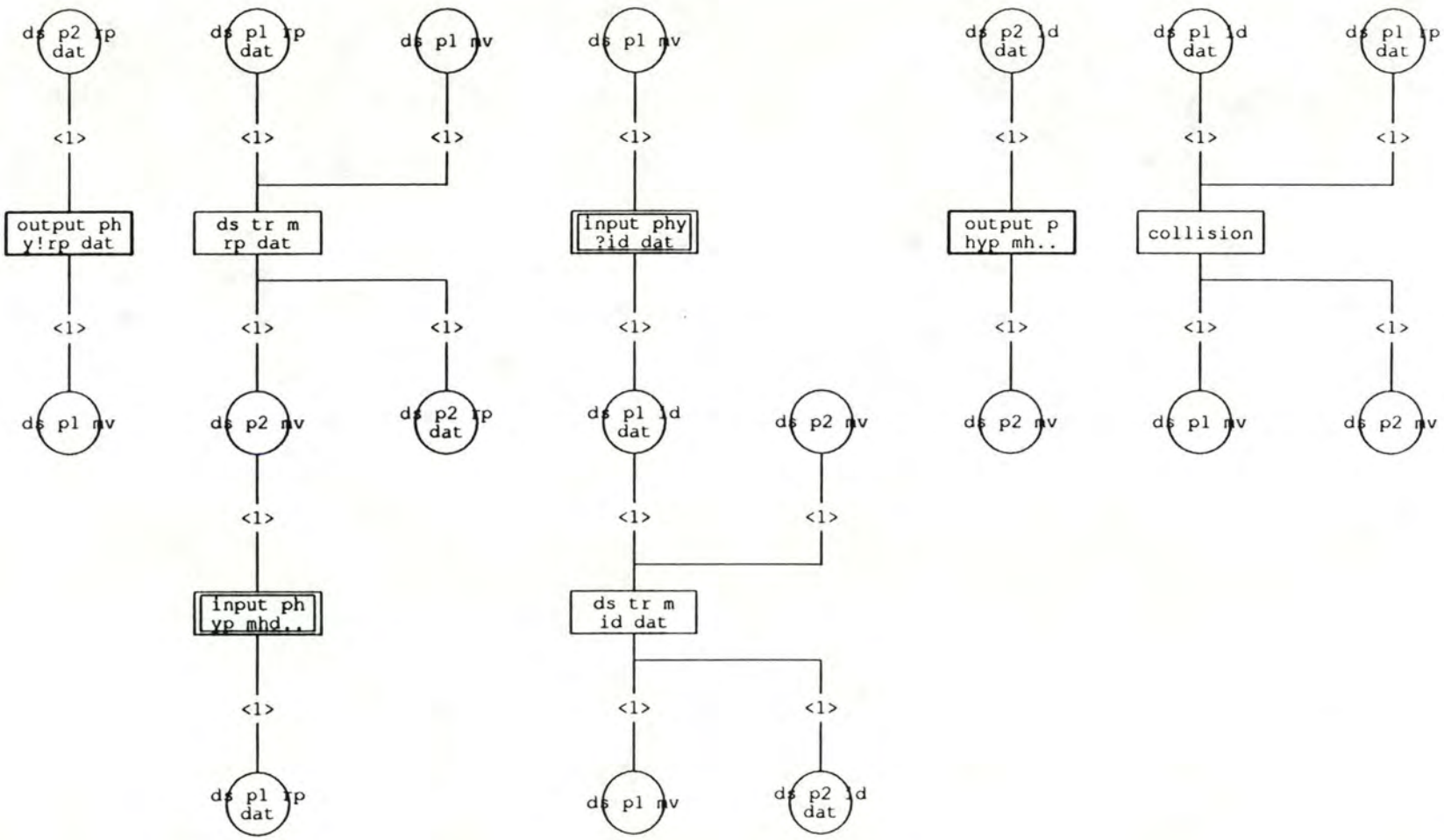


```
PLACE : wait_rp_dat
     MARKING :   0

PLACE : next_id_dat
     MARKING :   1

TRANSITION : erreur_rp_dat
     COMMUNICATION PARAMETERS : output_phy INPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : next_id_dat(1)
     POST CONDITIONS : next_id_dat(1)
TRANSITION : send_id_dat_per
     COMMUNICATION PARAMETERS : output_phy OUTPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : next_id_dat(1)
     POST CONDITIONS : wait_rp_dat(1)
TRANSITION : receive_rp_dat
     COMMUNICATION PARAMETERS : output_phy INPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : wait_rp_dat(1)
     POST CONDITIONS : next_id_dat(1)
TRANSITION : t1
     TIME : [16,18]
     PRE CONDITIONS : wait_rp_dat(1)
     POST CONDITIONS : next_id_dat(1)
 CONNECTIONS : ENV , output_phy
 EXTERNAL SYMBOLS : ds_tr_g_rp_dat
```

	DESCRIPTION: fip_per TYPE: time	Page: 6
1.0	BLOCK fip_per/mhdds	26-Dec-1989



	DESCRIPTION: fip_per TYPE: time	Page: 7
1.0	PETRI NET fip_per/mhdds/pn_mhdds	26-Dec-1989

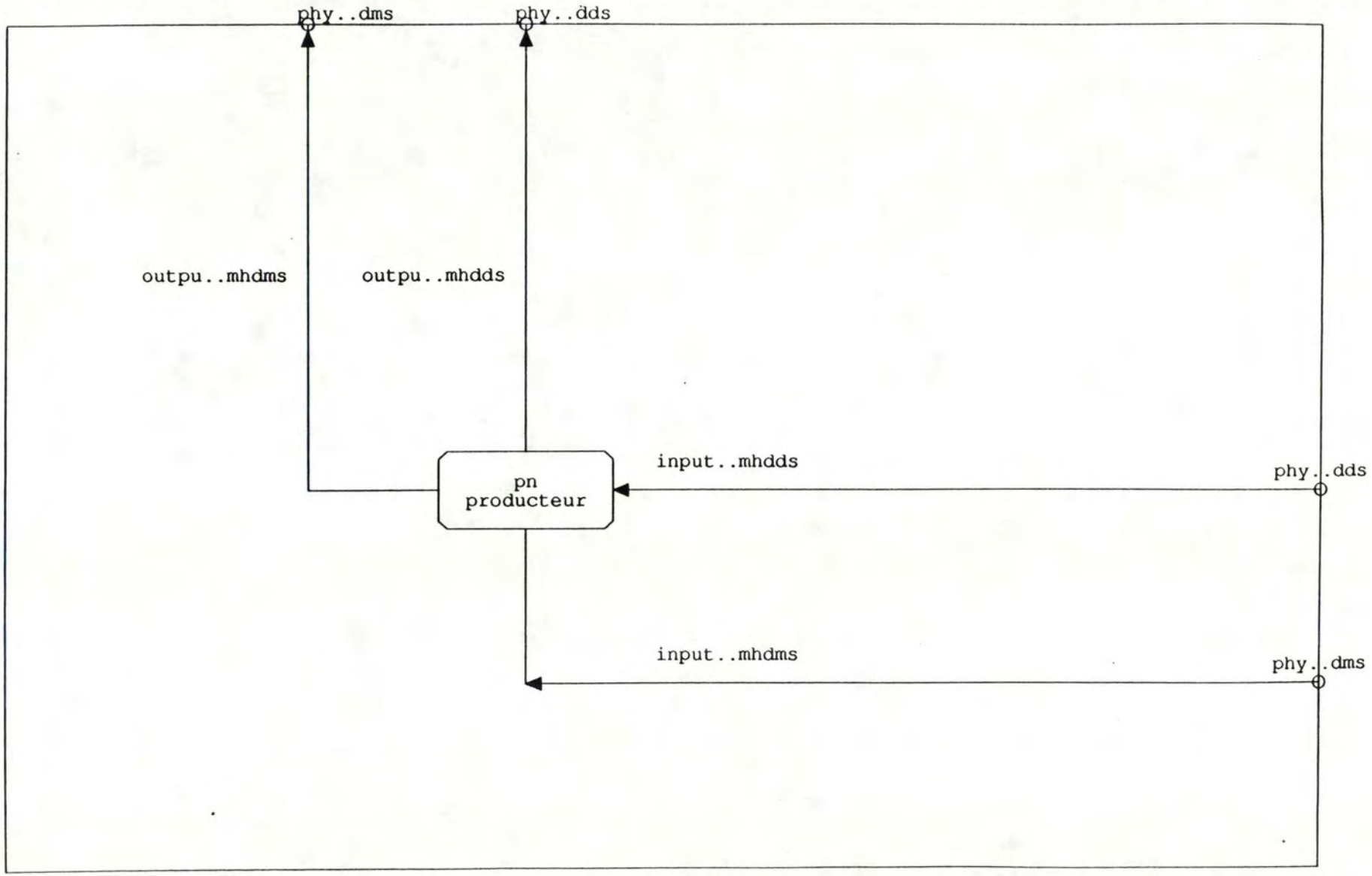


```
PLACE : ds_p2_id_dat
     MARKING :   0

PLACE : ds_p2_mv
     MARKING :   1

PLACE : ds_p1_id_dat
     MARKING :   0

PLACE : ds_p1_mv
     MARKING :   1

PLACE : ds_p1_rp_dat
     MARKING :   0

PLACE : ds_p2_rp_dat
     MARKING :   0

TRANSITION : ds_tr_g_rp_dat
     COMMUNICATION PARAMETERS : output_phy OUTPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : ds_p2_rp_dat(1)
     POST CONDITIONS : ds_p1_mv(1)
TRANSITION : ds_tr_m_rp_dat
     TIME : [1,1]
     PRE CONDITIONS : ds_p1_rp_dat(1)  , ds_p1_mv(1)
     POST CONDITIONS : ds_p2_mv(1)  , ds_p2_rp_dat(1)
TRANSITION : ds_tr_d_rp_dat
     COMMUNICATION PARAMETERS : output_phyp_mhdds INPUT rp_dat
     TIME : [9,11]
     PRE CONDITIONS : ds_p2_mv(1)
     POST CONDITIONS : ds_p1_rp_dat(1)
TRANSITION : ds_tr_g_id_dat
```

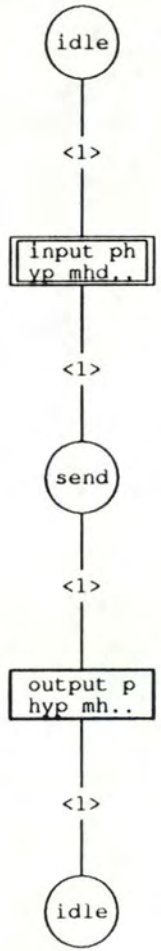
	DESCRIPTION: fip_per TYPE: time	Page: 9
1.0	PETRI NET fip_per/mhdds/pn_mhdds	26-Dec-1989

```
     COMMUNICATION PARAMETERS : output_phy INPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : ds_p1_mv(1)
     POST CONDITIONS : ds_p1_id_dat(1)
TRANSITION : ds_tr_m_id_dat
     TIME : [1,1]
     PRE CONDITIONS : ds_p1_id_dat(1)  , ds_p2_mv(1)
     POST CONDITIONS : ds_p1_mv(1)  , ds_p2_id_dat(1)
TRANSITION : ds_tr_d_id_dat
     COMMUNICATION PARAMETERS : output_inter_med OUTPUT id_dat , output_phyp_mhdds OUTPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : ds_p2_id_dat(1)
     POST CONDITIONS : ds_p2_mv(1)
TRANSITION : collision
     TIME : [0,0]
     PRE CONDITIONS : ds_p1_id_dat(1)  , ds_p1_rp_dat(1)
     POST CONDITIONS : ds_p1_mv(1)  , ds_p2_mv(1)
 CONNECTIONS : ENV , output_inter_med , output_phyp_mhdds , output_phy
 EXTERNAL SYMBOLS : send_rp_dat , send_id_dat_per
```

	DESCRIPTION: fip_per TYPE: time	Page: 10
1.0	BLOCK fip_per/producteur	26-Dec-1989



	DESCRIPTION: fip_per TYPE: time	Page: 11
1.0	PETRI NET fip_per/producteur/pn_producteur	26-Dec-1989



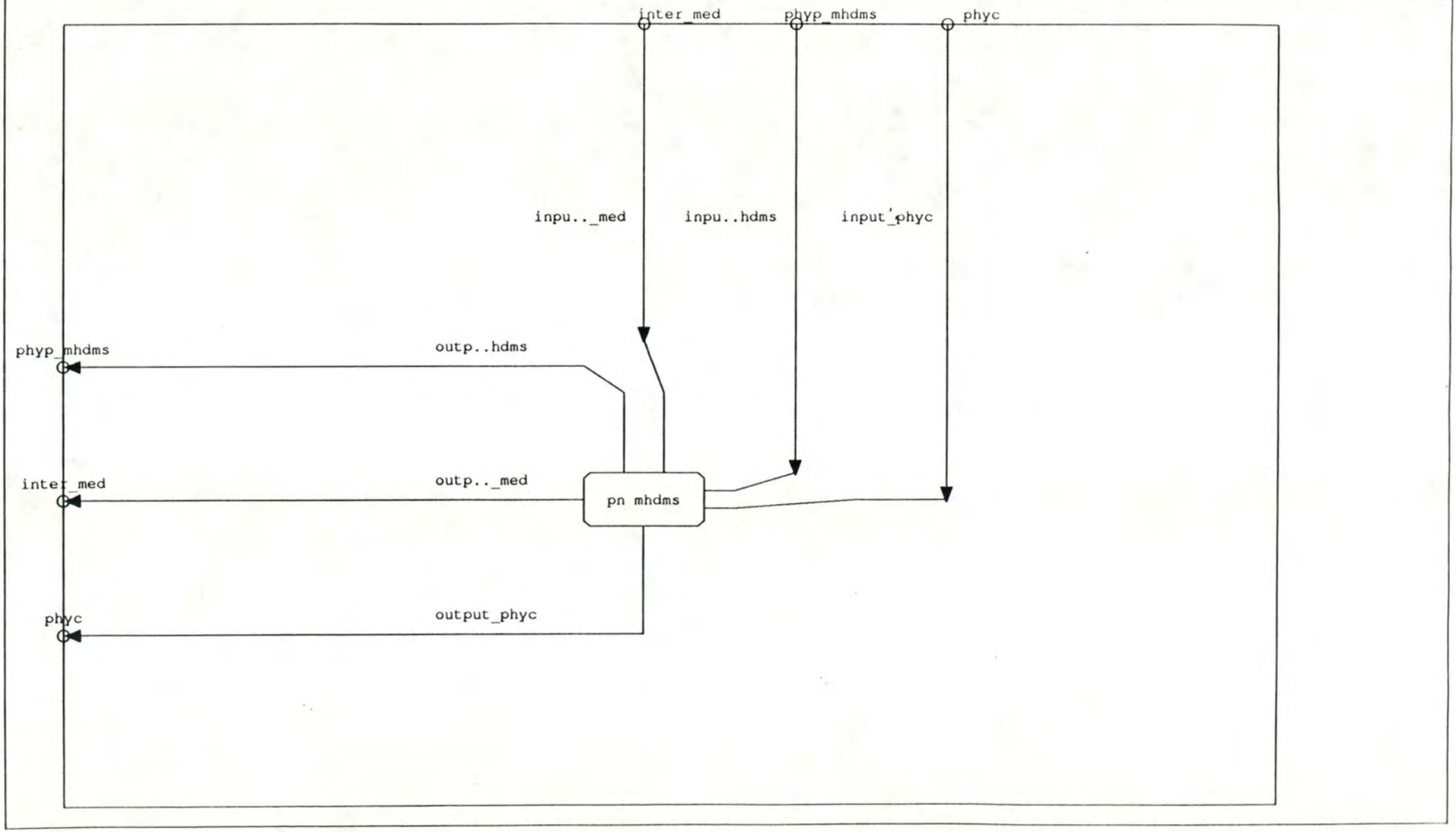
⊕ out..dms

⊕ out..dds

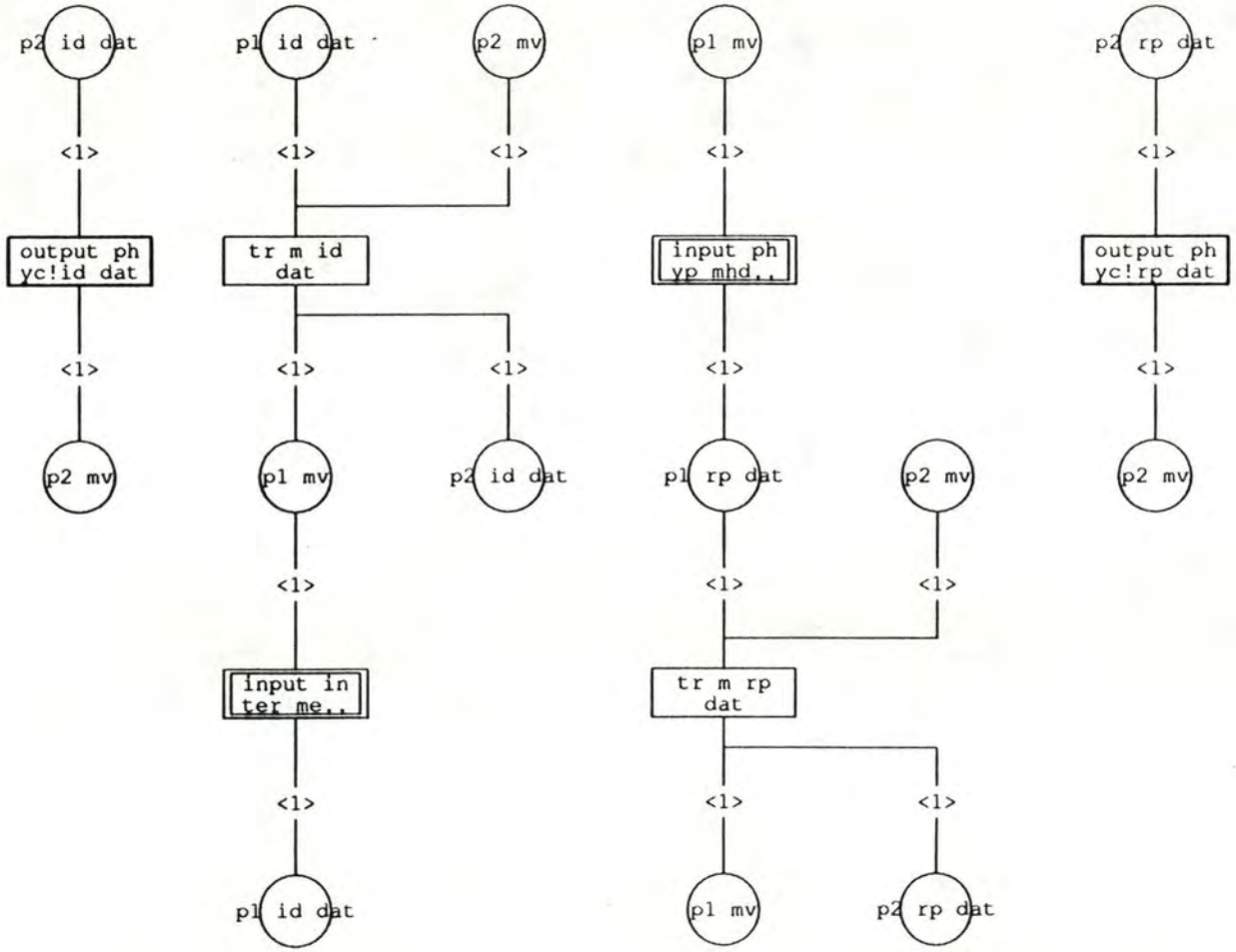
inp..dat

```
PLACE : send
     MARKING :    0

PLACE : idle
     MARKING :    1

TRANSITION : prepare_to_send
     COMMUNICATION PARAMETERS : output_phyp_mhdds INPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : idle(1)
     POST CONDITIONS : send(1)
TRANSITION : send_rp_dat
     COMMUNICATION PARAMETERS : output_phyp_mhdds OUTPUT rp_dat , output_phyp_mhdms OUTPUT rp_dat
     TIME : [9,11]
     PRE CONDITIONS : send(1)
     POST CONDITIONS : idle(1)
 CONNECTIONS : ENV , output_phyp_mhdms , output_phyp_mhdds
 EXTERNAL SYMBOLS : ds_tr_d_id_dat
```

	DESCRIPTION: fip_per TYPE: time	Page: 13
1.0	BLOCK fip_per/mhdms	26-Dec-1989



1.0	DESCRIPTION: fip_per TYPE: time	Page: 14
	PETRI NET fip_per/mhdms/pn_mhdms	26-Dec-1989



⊕ out..hyc

⊕ out..dms

⊕ out..med

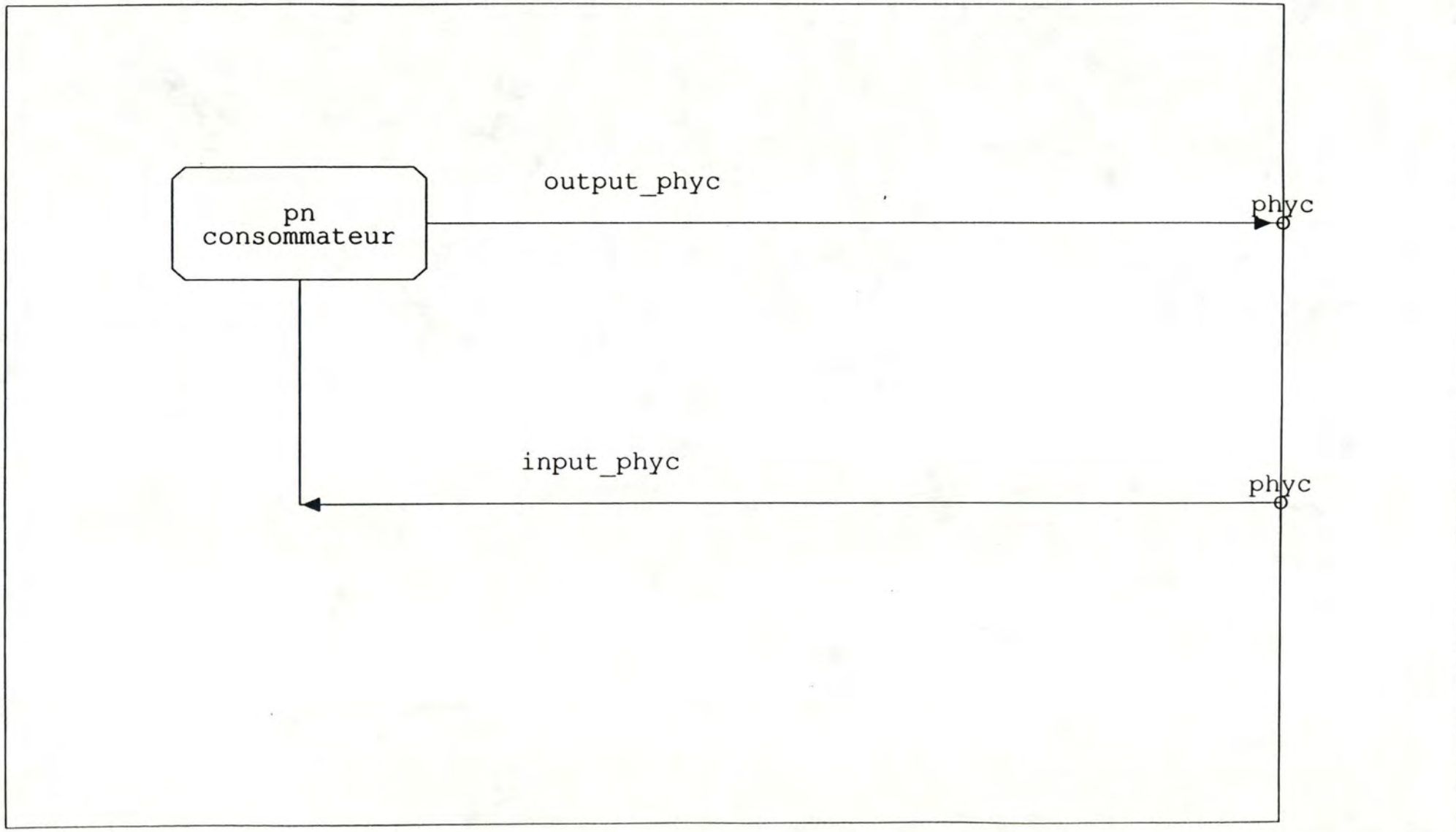
inp..dat

inp..dat

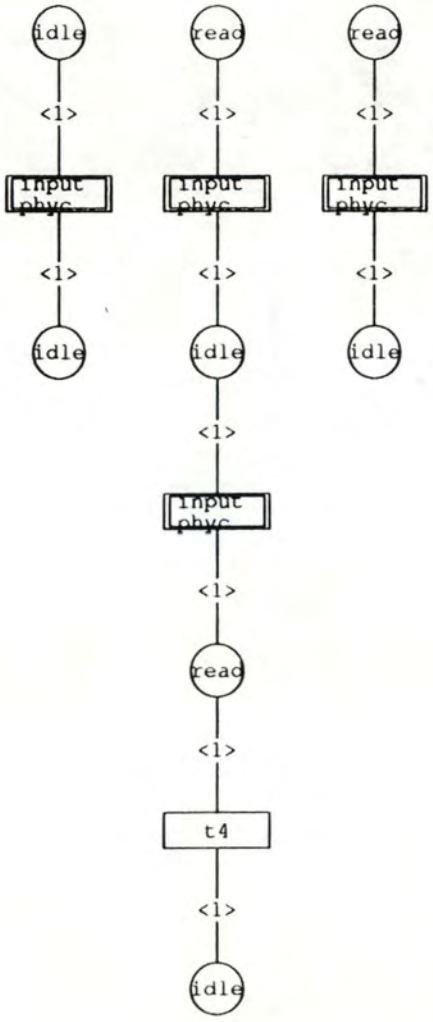
```
PLACE : p2_rp_dat
     MARKING :   0

PLACE : p1_rp_dat
     MARKING :   0

PLACE : p1_mv
     MARKING :   1

PLACE : p1_id_dat
     MARKING :   0

PLACE : p2_mv
     MARKING :   1

PLACE : p2_id_dat
     MARKING :   0

TRANSITION : tr_d_id_dat
     COMMUNICATION PARAMETERS : output_phyc OUTPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : p2_id_dat(1)
     POST CONDITIONS : p2_mv(1)
TRANSITION : tr_m_id_dat
     TIME : [1,1]
     PRE CONDITIONS : p1_id_dat(1)  , p2_mv(1)
     POST CONDITIONS : p1_mv(1)  , p2_id_dat(1)
TRANSITION : tr_g_id_dat
     COMMUNICATION PARAMETERS : output_inter_med INPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : p1_mv(1)
     POST CONDITIONS : p1_id_dat(1)
TRANSITION : tr_g_rp_dat
```

```
     COMMUNICATION PARAMETERS : output_phyp_mhdms INPUT rp_dat
     TIME : [9,11]
     PRE CONDITIONS : p1_mv(1)
     POST CONDITIONS : p1_rp_dat(1)
TRANSITION : tr_m_rp_dat
     TIME : [1,1]
     PRE CONDITIONS : p1_rp_dat(1)  , p2_mv(1)
     POST CONDITIONS : p1_mv(1)  , p2_rp_dat(1)
TRANSITION : tr_d_rp_dat
     COMMUNICATION PARAMETERS : output_phyc OUTPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : p2_rp_dat(1)
     POST CONDITIONS : p2_mv(1)
 CONNECTIONS : ENV , output_phyc , output_phyp_mhdms , output_inter_med
 EXTERNAL SYMBOLS : send_rp_dat , ds_tr_d_id_dat
```

	DESCRIPTION: fip_per TYPE: time	Page: 17
1.0	BLOCK fip_per/consommateur	26-Dec-1989



	DESCRIPTION: fip_per TYPE: time	Page: 18
1.0	PETRI NET fip_per/consommateur/pn_consommateur	26-Dec-1989



⊕ out..hyc

inp..dat

inp..dat

```
PLACE : idle
     MARKING :   1

PLACE : read
     MARKING :   0

TRANSITION : erreur_rp_dat
     COMMUNICATION PARAMETERS : output_phyc INPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : idle(1)
     POST CONDITIONS : idle(1)
TRANSITION : read_rp_dat
     COMMUNICATION PARAMETERS : output_phyc INPUT rp_dat
     TIME : [0,0]
     PRE CONDITIONS : read(1)
     POST CONDITIONS : idle(1)
TRANSITION : prepare_to_read
     COMMUNICATION PARAMETERS : output_phyc INPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : idle(1)
     POST CONDITIONS : read(1)
TRANSITION : t4
     TIME : [13,15]
     PRE CONDITIONS : read(1)
     POST CONDITIONS : idle(1)
TRANSITION : erreur_id_dat
     COMMUNICATION PARAMETERS : output_phyc INPUT id_dat
     TIME : [0,0]
     PRE CONDITIONS : read(1)
     POST CONDITIONS : idle(1)
 CONNECTIONS : ENV , output_phyc
 EXTERNAL SYMBOLS : tr_d_id_dat , tr_d_rp_dat
```

ANNEXE 3

MODELE FIP_PER - RESULTATS DE L'ANALYSE

Fonctionnement du système sans erreur

Results of the Timed Analysis

ENUMERATION IS TERMINATED

BOUNDED 16 STATE CLASSES

*** CPU TIME = 1 SEC ***

NOT LIVE

CYCLIC

*** 0 TOTAL DEADLOCK CLASS ***

--- UPPER BOUND FOR PLACES MARKING

mhdds__pn_mhdds__ds_p2_mv(1)

mhdms__pn_mhdms__p1_mv(1)

producteur__pn_producteur__send(1)

mhdds__pn_mhdds__ds_p1_rp_dat(1)

mhdms__pn_mhdms__p1_rp_dat(1)

producteur__pn_producteur__idle(1)

mhdds__pn_mhdds__ds_p2_id_dat(1)

mhdms__pn_mhdms__p1_id_dat(1)

mhdds__pn_mhdds__ds_p1_mv(1)

mhdds__pn_mhdds__ds_p2_rp_dat(1)

mhdds__pn_mhdds__ds_p1_id_dat(1)

mhdms__pn_mhdms__p2_rp_dat(1)

consommateur__pn_consommateur__idle(1)

mhdms__pn_mhdms__p2_mv(1)

mhdms__pn_mhdms__p2_id_dat(1)

consommateur__pn_consommateur__read(1)

arbitre__pn_arbitre__wait_rp_dat(1)

arbitre__pn_arbitre__next_id_dat(1)

TR1 = /consommateur__pn_consommateur__erreur_id_dat || /mhdms__pn_mhdms__tr_d_id_dat

TR2 = /consommateur__pn_consommateur__read_rp_dat || /mhdms__pn_mhdms__tr_d_rp_dat

TR3 = /consommateur__pn_consommateur__prepare_to_read || /mhdms__pn_mhdms__tr_d_id_-
dat

TR4 = /consommateur__pn_consommateur__erreur_rp_dat || /mhdms__pn_mhdms__tr_d_rp_dat

TR5 = /mhdms__pn_mhdms__tr_g_rp_dat || /producteur__pn_producteur__send_rp_dat

TR6 = /producteur__pn_producteur__prepare_to_send || /mhdms__pn_mhdms__tr_g_id_dat -
|| /mhdds__pn_mhdds__ds_tr_d_id_dat

TR7 = /TR5 || /mhdds__pn_mhdds__ds_tr_d_rp_dat

TR8 = /mhdds__pn_mhdds__ds_tr_g_rp_dat || /arbitre__pn_arbitre__erreur_rp_dat

TR9 = /mhdds__pn_mhdds__ds_tr_g_rp_dat || /arbitre__pn_arbitre__receive_rp_dat

TR10 = /mhdds__pn_mhdds__ds_tr_g_id_dat || /arbitre__pn_arbitre__send_id_dat_per

% NET DEFINITION : fip_per

% 18 PLACES, 16 TRANSITIONS

TR10[0,0]: arbitre__pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv -> arbitre__pn-
_arbitre__wait_rp_dat, mhdds__pn_mhdds__ds_p1_id_dat

TR9[0,0]: arbitre__pn_arbitre__wait_rp_dat, mhdds__pn_mhdds__ds_p2_rp_dat -> arbitre_-
_pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv

TR8[0,0]: arbitre__pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p2_rp_dat -> arbitre_-
_pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv

arbitre__pn_arbitre__t1[16,18]: arbitre__pn_arbitre__wait_rp_dat -> arbitre__pn_arbit-
re__next_id_dat

consommateur__pn_consommateur__t4[13,15]: consommateur__pn_consommateur__read -> cons-
ommateur__pn_consommateur__idle

mhdms__pn_mhdms__tr_m_rp_dat[1,1]: mhdms__pn_mhdms__p2_mv, mhdms__pn_mhdms__p1_rp_dat-
-> mhdms__pn_mhdms__p2_rp_dat, mhdms__pn_mhdms__p1_mv

mhdms__pn_mhdms__tr_m_id_dat[1,1]: mhdms__pn_mhdms__p2_mv, mhdms__pn_mhdms__p1_id_dat-
-> mhdms__pn_mhdms__p2_id_dat, mhdms__pn_mhdms__p1_mv

TR1[0,0]: consommateur__pn_consommateur__read, mhdms__pn_mhdms__p2_id_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

TR2[0,0]: consommateur__pn_consommateur__read, mhdms__pn_mhdms__p2_rp_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

TR3[0,0]: mhdms__pn_mhdms__p2_id_dat, consommateur__pn_consommateur__idle -> consomma-
teur__pn_consommateur__read, mhdms__pn_mhdms__p2_mv

TR4[0,0]: consommateur__pn_consommateur__idle, mhdms__pn_mhdms__p2_rp_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

mhdds__pn_mhdds__collision[0,0]: mhdds__pn_mhdds__ds_p1_id_dat, mhdds__pn_mhdds__ds_p-
1_rp_dat -> mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_p2_mv

mhdds__pn_mhdds__ds_tr_m_id_dat[1,1]: mhdds__pn_mhdds__ds_p1_id_dat, mhdds__pn_mhdds_-
_ds_p2_mv -> mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_p2_id_dat

mhdds__pn_mhdds__ds_tr_m_rp_dat[1,1]: mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_-
p1_rp_dat -> mhdds__pn_mhdds__ds_p2_rp_dat, mhdds__pn_mhdds__ds_p2_mv

TR6[0,0]: mhdds__pn_mhdds__ds_p2_id_dat, producteur__pn_producteur__idle, mhdms__pn_m-
hdms__p1_mv -> mhdms__pn_mhdms__p1_id_dat, producteur__pn_producteur__send, mhdds__pn_-

```
        mhdds__ds_p2_mv

TR7[9,11]: producteur__pn_producteur__send, mhdms__pn_mhdms__p1_mv, mhdds__pn_mhdds__-
        ds_p2_mv -> producteur__pn_producteur__idle, mhdms__pn_mhdms__p1_rp_dat, mhdds__pn_mhd-
        ds__ds_p1_rp_dat

% INITIAL PLACE MARKING

M0 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producte-
        ur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhd-
        ms__pn_mhdms__p2_mv(1), arbitre__pn_arbitre__next_id_dat(1)
```

classes increment is: 2250

CPU time increment is: 32768

specific condition is: EMPTY

C0 -> (TR10:[0,0])/C1

C1 -> (mhdds__pn_mhdds__ds_tr_m_id_dat:[1,1])/C2

C2 -> (TR6:[0,0])/C3

C3 -> (mhdms__pn_mhdms__tr_m_id_dat:[1,1])/C4

C4 -> (TR3:[0,0])/C5

C5 -> (TR7:[9,11])/C6

C6 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[1,1])/C7,

(mhdms__pn_mhdms__tr_m_rp_dat:[1,1])/C14

C7 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C8, (TR9:[0,0])/C12

C8 -> (TR2:[0,0])/C9, (TR9:[0,0])/C10

C9 -> (TR9:[0,0])/C0

C10 -> (TR2:[0,0])/C0, (TR10:[0,0])/C11

C11 -> (TR2:[0,0])/C1

C12 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C10, (TR10:[0,0])/C13

C13 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C11

C14 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C8, (TR2:[0,0])/C15

C15 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C9

CLASS *C0: *M0 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__next_id_dat(1)
*D0 = 0 <= TR10 <= 0;

CLASS *C1: *M1 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D1 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1;
16 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C2: *M2 = mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__idle(1),
mhdds__pn_mhdds__ds_p2_id_dat(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D2 = 0 <= TR6 <= 0; 15 <= arbitre__pn_arbitre__t1 <= 17;

CLASS *C3: *M3 = mhdds__pn_mhdds__ds_p2_mv(1), producteur__pn_producteur__send(1), mhdms__pn_mhdms__p1_id_dat(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D3 = 1 <= mhdms__pn_mhdms__tr_m_id_dat <= 1;

15 <= arbitre__pn_arbitre__t1 <= 17;

CLASS *C4: *M4 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__send(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_id_dat(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D4 = 9 <= TR7 <= 11; 0 <= TR3 <= 0; 14 <= arbitre__pn_arbitre__t1 <= 16;

CLASS *C5: *M5 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__send(1), mhdds__pn_mhdds__ds_p1_mv(1), mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D5 = 9 <= TR7 <= 11; 13 <= consommateur__pn_consommateur__t4 <= 15;

14 <= arbitre__pn_arbitre__t1 <= 16;

CLASS *C6: *M6 = mhdds__pn_mhdds__ds_p1_rp_dat(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D6 = 1 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 1;

1 <= mhdms__pn_mhdms__tr_m_rp_dat <= 1; 2 <= consommateur__pn_consommateur__t4 <= 6;

3 <= arbitre__pn_arbitre__t1 <= 7;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 1;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 3;

CLASS *C7: *M7 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D7 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0;

1 <= consommateur__pn_consommateur__t4 <= 5; 2 <= arbitre__pn_arbitre__t1 <= 6;

0 <= TR9 <= 0;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 1;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 3;

CLASS *C8: *M8 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D8 = 0 <= TR2 <= 0; 1 <= consommateur__pn_consommateur__t4 <= 5;
2 <= arbitre__pn_arbitre__t1 <= 6; 0 <= TR9 <= 0;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 1;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 3;

CLASS *C9: *M9 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D9 = 2 <= arbitre__pn_arbitre__t1 <= 6; 0 <= TR9 <= 0;

CLASS *C10: *M10 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__next_id_dat(1)
*D10 = 0 <= TR2 <= 0; 1 <= consommateur__pn_consommateur__t4 <= 5;
0 <= TR10 <= 0;
CLASS *C11: *M11 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D11 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1; 0 <= TR2 <= 0;
1 <= consommateur__pn_consommateur__t4 <= 5; 16 <= arbitre__pn_arbitre__t1 <= 18;
CLASS *C12: *M12 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__next_id_dat(1)

*D12 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0;

1 <= consommateur__pn_consommateur__t4 <= 5; 0 <= TR10 <= 0;

CLASS *C13: *M13 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),

mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D13 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1;

0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0; 1 <= consommateur__pn_consommateur__t4 <= 5;

16 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C14: *M14 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),

mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D14 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0; 0 <= TR2 <= 0;

1 <= consommateur__pn_consommateur__t4 <= 5; 2 <= arbitre__pn_arbitre__t1 <= 6;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 1;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 3;

CLASS *C15: *M15 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),

consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D15 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0;

2 <= arbitre__pn_arbitre__t1 <= 6;

CFCMS CONTENTS :

*G0 = {C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15}

CFCMS GRAPH :

G0 -> TR7/G0, TR6/G0, mhdds__pn_mhdds__ds_tr_m_rp_dat/G0,

mhdds__pn_mhdds__ds_tr_m_id_dat/G0, TR3/G0, TR2/G0, mhdms__pn_mhdms__tr_m_id_dat/G0,

mhdms__pn_mhdms__tr_m_rp_dat/G0, TR9/G0, TR10/G0

ANNEXE 4

MODELE FIP_PER - RESULTATS DE L'ANALYSE

Fonctionnement du système avec erreurs

Results of the Timed Analysis

ENUMERATION IS TERMINATED

BOUNDED 26 STATE CLASSES

*** CPU TIME = 2 SEC ***

NOT LIVE

CYCLIC

*** 0 TOTAL DEADLOCK CLASS ***

--- UPPER BOUND FOR PLACES MARKING

mhdds__pn_mhdds__ds_p2_mv(1)

mhdms__pn_mhdms__p1_mv(1)

producteur__pn_producteur__send(1)

mhdds__pn_mhdds__ds_p1_rp_dat(1)

mhdms__pn_mhdms__p1_rp_dat(1)

producteur__pn_producteur__idle(1)

mhdds__pn_mhdds__ds_p2_id_dat(1)

mhdms__pn_mhdms__p1_id_dat(1)

mhdds__pn_mhdds__ds_p1_mv(1)

mhdds__pn_mhdds__ds_p2_rp_dat(1)

mhdds__pn_mhdds__ds_p1_id_dat(1)

mhdms__pn_mhdms__p2_rp_dat(1)

consommateur__pn_consommateur__idle(1)

mhdms__pn_mhdms__p2_mv(1)

mhdms__pn_mhdms__p2_id_dat(1)

consommateur__pn_consommateur__read(1)

arbitre__pn_arbitre__wait_rp_dat(1)

arbitre__pn_arbitre__next_id_dat(1)

TR1 = /consommateur__pn_consommateur__erreur_id_dat || /mhdms__pn_mhdms__tr_d_id_dat

TR2 = /consommateur__pn_consommateur__read_rp_dat || /mhdms__pn_mhdms__tr_d_rp_dat

TR3 = /consommateur__pn_consommateur__prepare_to_read || /mhdms__pn_mhdms__tr_d_id_-
dat

TR4 = /consommateur__pn_consommateur__erreur_rp_dat || /mhdms__pn_mhdms__tr_d_rp_dat

TR5 = /mhdms__pn_mhdms__tr_g_rp_dat || /producteur__pn_producteur__send_rp_dat

TR6 = /producteur__pn_producteur__prepare_to_send || /mhdms__pn_mhdms__tr_g_id_dat -
|| /mhdds__pn_mhdds__ds_tr_d_id_dat

TR7 = /TR5 || /mhdds__pn_mhdds__ds_tr_d_rp_dat

TR8 = /mhdds__pn_mhdds__ds_tr_g_rp_dat || /arbitre__pn_arbitre__erreur_rp_dat

TR9 = /mhdds__pn_mhdds__ds_tr_g_rp_dat || /arbitre__pn_arbitre__receive_rp_dat

TR10 = /mhdds__pn_mhdds__ds_tr_g_id_dat || /arbitre__pn_arbitre__send_id_dat_per

% NET DEFINITION : fip_per

% 18 PLACES, 16 TRANSITIONS

TR10[0,0]: arbitre__pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv -> arbitre__pn-
_arbitre__wait_rp_dat, mhdds__pn_mhdds__ds_p1_id_dat

TR9[0,0]: arbitre__pn_arbitre__wait_rp_dat, mhdds__pn_mhdds__ds_p2_rp_dat -> arbitre_-
_pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv

TR8[0,0]: arbitre__pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p2_rp_dat -> arbitre_-
_pn_arbitre__next_id_dat, mhdds__pn_mhdds__ds_p1_mv

arbitre__pn_arbitre__t1[15,18]: arbitre__pn_arbitre__wait_rp_dat -> arbitre__pn_arbit-
re__next_id_dat

consommateur__pn_consommateur__t4[12,15]: consommateur__pn_consommateur__read -> cons-
ommateur__pn_consommateur__idle

mhdms__pn_mhdms__tr_m_rp_dat[1,1]: mhdms__pn_mhdms__p2_mv, mhdms__pn_mhdms__p1_rp_dat-
-> mhdms__pn_mhdms__p2_rp_dat, mhdms__pn_mhdms__p1_mv

mhdms__pn_mhdms__tr_m_id_dat[1,1]: mhdms__pn_mhdms__p2_mv, mhdms__pn_mhdms__p1_id_dat-
-> mhdms__pn_mhdms__p2_id_dat, mhdms__pn_mhdms__p1_mv

TR1[0,0]: consommateur__pn_consommateur__read, mhdms__pn_mhdms__p2_id_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

TR2[0,0]: consommateur__pn_consommateur__read, mhdms__pn_mhdms__p2_rp_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

TR3[0,0]: mhdms__pn_mhdms__p2_id_dat, consommateur__pn_consommateur__idle -> consomma-
teur__pn_consommateur__read, mhdms__pn_mhdms__p2_mv

TR4[0,0]: consommateur__pn_consommateur__idle, mhdms__pn_mhdms__p2_rp_dat -> mhdms__p-
n_mhdms__p2_mv, consommateur__pn_consommateur__idle

mhdds__pn_mhdds__collision[0,0]: mhdds__pn_mhdds__ds_p1_id_dat, mhdds__pn_mhdds__ds_p-
1_rp_dat -> mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_p2_mv

mhdds__pn_mhdds__ds_tr_m_id_dat[1,1]: mhdds__pn_mhdds__ds_p1_id_dat, mhdds__pn_mhdds_-
_ds_p2_mv -> mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_p2_id_dat

mhdds__pn_mhdds__ds_tr_m_rp_dat[1,1]: mhdds__pn_mhdds__ds_p1_mv, mhdds__pn_mhdds__ds_-
p1_rp_dat -> mhdds__pn_mhdds__ds_p2_rp_dat, mhdds__pn_mhdds__ds_p2_mv

TR6[0,0]: mhdds__pn_mhdds__ds_p2_id_dat, producteur__pn_producteur__idle, mhdms__pn_m-
hdms__p1_mv -> mhdms__pn_mhdms__p1_id_dat, producteur__pn_producteur__send, mhdds__pn_-

```
        mhdds__ds_p2_mv

TR7[9,11]: producteur__pn_producteur__send, mhdms__pn_mhdms__p1_mv, mhdds__pn_mhdds__-
        ds_p2_mv -> producteur__pn_producteur__idle, mhdms__pn_mhdms__p1_rp_dat, mhdds__pn_mhd-
        ds__ds_p1_rp_dat

% INITIAL PLACE MARKING

M0 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producte-
        ur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhd-
        ms__pn_mhdms__p2_mv(1), arbitre__pn_arbitre__next_id_dat(1)
```

classes increment is: 2250

CPU time increment is: 32768

specific condition is: EMPTY

C0 -> (TR10:[0,0])/C1

C1 -> (mhdds__pn_mhdds__ds_tr_m_id_dat:[1,1])/C2

C2 -> (TR6:[0,0])/C3

C3 -> (mhdms__pn_mhdms__tr_m_id_dat:[1,1])/C4

C4 -> (TR3:[0,0])/C5

C5 -> (TR7:[9,11])/C6

C6 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[1,1])/C7,
(mhdms__pn_mhdms__tr_m_rp_dat:[1,1])/C21, (consommateur__pn_consommateur__t4:[1,1])/C25

C7 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C8,
(consommateur__pn_consommateur__t4:[0,0])/C16, (TR9:[0,0])/C19

C8 -> (TR2:[0,0])/C9, (consommateur__pn_consommateur__t4:[0,0])/C10, (TR9:[0,0])/C14

C9 -> (TR9:[0,0])/C0

C10 -> (TR4:[0,0])/C11, (TR9:[0,0])/C12

C11 -> (TR9:[0,0])/C0

C12 -> (TR4:[0,0])/C0, (TR10:[0,0])/C13

C13 -> (TR4:[0,0])/C1

C14 -> (TR2:[0,0])/C0, (consommateur__pn_consommateur__t4:[0,0])/C12, (TR10:[0,0])/C15

C15 -> (TR2:[0,0])/C1, (consommateur__pn_consommateur__t4:[0,0])/C13

C16 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C10, (TR9:[0,0])/C17

C17 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C12, (TR10:[0,0])/C18

C18 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C13

C19 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C14,

(consommateur__pn_consommateur__t4:[0,0])/C17, (TR10:[0,0])/C20

C20 -> (mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C15,

(consommateur__pn_consommateur__t4:[0,0])/C18

C21 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C8, (TR2:[0,0])/C22,

(consommateur__pn_consommateur__t4:[0,0])/C23

C22 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C9

C23 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C10, (TR4:[0,0])/C24

C24 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C11

C25 -> (mhdds__pn_mhdds__ds_tr_m_rp_dat:[0,0])/C16,

(mhdms__pn_mhdms__tr_m_rp_dat:[0,0])/C23

CLASS *C0: *M0 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__next_id_dat(1)
*D0 = 0 <= TR10 <= 0;
CLASS *C1: *M1 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D1 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1;
15 <= arbitre__pn_arbitre__t1 <= 18;
CLASS *C2: *M2 = mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__idle(1),
mhdds__pn_mhdds__ds_p2_id_dat(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D2 = 0 <= TR6 <= 0; 14 <= arbitre__pn_arbitre__t1 <= 17;

CLASS *C3: *M3 = mhdds__pn_mhdds__ds_p2_mv(1), producteur__pn_producteur__send(1), mhdms__pn_mhdms__p1_id_dat(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D3 = 1 <= mhdms__pn_mhdms__tr_m_id_dat <= 1;
14 <= arbitre__pn_arbitre__t1 <= 17;

CLASS *C4: *M4 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__send(1), mhdds__pn_mhdds__ds_p1_mv(1), consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_id_dat(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D4 = 9 <= TR7 <= 11; 0 <= TR3 <= 0; 13 <= arbitre__pn_arbitre__t1 <= 16;

CLASS *C5: *M5 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1), producteur__pn_producteur__send(1), mhdds__pn_mhdds__ds_p1_mv(1), mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1), arbitre__pn_arbitre__wait_rp_dat(1)

*D5 = 9.<= TR7 <= 11; 12 <= consommateur__pn_consommateur__t4 <= 15;

13 <= arbitre__pn_arbitre__t1 <= 16;

CLASS *C6: *M6 = mhdds__pn_mhdds__ds_p1_rp_dat(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D6 = 1 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 1;

1 <= mhdms__pn_mhdms__tr_m_rp_dat <= 1; 1 <= consommateur__pn_consommateur__t4 <= 6;

2 <= arbitre__pn_arbitre__t1 <= 7;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 2;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 4;

CLASS *C7: *M7 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D7 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0;

0 <= consommateur__pn_consommateur__t4 <= 5; 1 <= arbitre__pn_arbitre__t1 <= 6;

0 <= TR9 <= 0;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 2;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 4;

CLASS *C8: *M8 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D8 = 0 <= TR2 <= 0; 0 <= consommateur__pn_consommateur__t4 <= 5;
1 <= arbitre__pn_arbitre__t1 <= 6; 0 <= TR9 <= 0;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 2;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 4;

CLASS *C9: *M9 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D9 = 1 <= arbitre__pn_arbitre__t1 <= 6; 0 <= TR9 <= 0;

CLASS *C10: *M10 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__idle(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D10 = 0 <= TR4 <= 0; 1 <= arbitre__pn_arbitre__t1 <= 4; 0 <= TR9 <= 0;
CLASS *C11: *M11 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D11 = 1 <= arbitre__pn_arbitre__t1 <= 4; 0 <= TR9 <= 0;
CLASS *C12: *M12 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__idle(1),
arbitre__pn_arbitre__next_id_dat(1)
*D12 = 0 <= TR4 <= 0; 0 <= TR10 <= 0;
CLASS *C13: *M13 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__idle(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D13 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1; 0 <= TR4 <= 0;
15 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C14: *M14 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__next_id_dat(1)
*D14 = 0 <= TR2 <= 0; 0 <= consommateur__pn_consommateur__t4 <= 5;
0 <= TR10 <= 0;

CLASS *C15: *M15 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_mv(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)
*D15 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1; 0 <= TR2 <= 0;

0 <= consommateur__pn_consommateur__t4 <= 5; 15 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C16: *M16 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p2_rp_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D16 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0;
1 <= arbitre__pn_arbitre__t1 <= 4; 0 <= TR9 <= 0;

CLASS *C17: *M17 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__next_id_dat(1)

*D17 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0; 0 <= TR10 <= 0;

CLASS *C18: *M18 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D18 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1;

0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0; 15 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C19: *M19 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__next_id_dat(1)

*D19 = 0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0;

0 <= consommateur__pn_consommateur__t4 <= 5; 0 <= TR10 <= 0;

CLASS *C20: *M20 = mhdds__pn_mhdds__ds_p2_mv(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_id_dat(1),
mhdms__pn_mhdms__p2_mv(1), consommateur__pn_consommateur__read(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D20 = 1 <= mhdds__pn_mhdds__ds_tr_m_id_dat <= 1;

0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0; 0 <= consommateur__pn_consommateur__t4 <= 5;
15 <= arbitre__pn_arbitre__t1 <= 18;

CLASS *C21: *M21 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),

mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__read(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D21 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0; 0 <= TR2 <= 0;

0 <= consommateur__pn_consommateur__t4 <= 5; 1 <= arbitre__pn_arbitre__t1 <= 6;

consommateur__pn_consommateur__t4 - arbitre__pn_arbitre__t1 <= 2;

arbitre__pn_arbitre__t1 - consommateur__pn_consommateur__t4 <= 4;

CLASS *C22: *M22 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),

consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D22 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0;

1 <= arbitre__pn_arbitre__t1 <= 6;

CLASS *C23: *M23 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),

producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),

mhdms__pn_mhdms__p2_rp_dat(1), consommateur__pn_consommateur__idle(1),

arbitre__pn_arbitre__wait_rp_dat(1)

*D23 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0; 0 <= TR4 <= 0;

1 <= arbitre__pn_arbitre__t1 <= 4;

CLASS *C24: *M24 = mhdms__pn_mhdms__p1_mv(1), mhdds__pn_mhdds__ds_p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D24 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0;

1 <= arbitre__pn_arbitre__t1 <= 4;

CLASS *C25: *M25 = mhdds__pn_mhdds__ds_p1_rp_dat(1), mhdms__pn_mhdms__p1_rp_dat(1),
producteur__pn_producteur__idle(1), mhdds__pn_mhdds__ds_p1_mv(1),
consommateur__pn_consommateur__idle(1), mhdms__pn_mhdms__p2_mv(1),
arbitre__pn_arbitre__wait_rp_dat(1)

*D25 = 0 <= mhdds__pn_mhdds__ds_tr_m_rp_dat <= 0;

0 <= mhdms__pn_mhdms__tr_m_rp_dat <= 0; 1 <= arbitre__pn_arbitre__t1 <= 4;

CFCMS CONTENTS :

*G0 = {C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17,
C18, C19, C20, C21, C22, C23, C24, C25}

CFCMS GRAPH :

G0 -> TR7/G0, TR6/G0, mhdds__pn_mhdds__ds_tr_m_rp_dat/G0,
mhdds__pn_mhdds__ds_tr_m_id_dat/G0, TR4/G0, TR3/G0, TR2/G0,
mhdms__pn_mhdms__tr_m_id_dat/G0, mhdms__pn_mhdms__tr_m_rp_dat/G0,
consommateur__pn_consommateur__t4/G0, TR9/G0, TR10/G0

Réalisation, impression et reliures : SPRL Centre de Reprographie - Tél. : 081/22.87.72